Gonny Hauwert

Time dependent optimization problems in networks

Master thesis, defended on November 10, 2010

Thesis advisors: Dr. F.M. Spieksma and Prof. Dr. K.I. Aardal

Specialisation: Applied Mathematics

Mathematisch Instituut, Universiteit Leiden

Contents

1. Introduction

A common phenomenon in the modern world is *traffic congestion*. Traffic congestion is a condition on road networks that occurs as use increases. It is characterized by longer travel times and increased vehicular queueing. This congestion is typically more pronounced in one direction in certain time intervals during the day; in the morning in one direction and in the late afternoon in the other. Most car drivers like to know before they leave "which road should I take to get to my destination as fast as possible?". Also, for distribution companies is this question important in order to deliver products fast and as cheap as possible.
The question of finding the shortest path or route for given static travel times has been a central problem in optimization for several decades. For the situation that the travel times changes over time less is known. The topic of this thesis is to investigate how various routing problems are affected by making the travel time dynamic. More specifically, we consider time dependent algorithms for different optimization problems.
The *vehicle routing problem* (VRP) is the problem to determine an amount of K vehicle routes, where a route is a tour that begins at the depot, traverses a subset of the cities in a specified sequence and returns to the depot. Each city must be assigned to exactly one of the K vehicle routes and the total size of deliveries for cities assigned to each vehicle must not exceed the vehicle capacity. The routes should be chosen such as to minimize the total travel distance. Until recently the vehicle routing models relied on a constant travel speed on every road during the whole day, but this is far from reality because of the rush hours. Recently, the interest in the time dependent vehicle routing models which take time into account, such that the model can handle different travel times on the same road, has increased. This version of the VRP models the traffic situations where congestions play a role in a more realistic way.
A special case of the VRP is called the *traveling salesman problem* (TSP), which is the case when we only have one vehicle in the VRP. Given a set of cities and the cost of travel between each pair of them, the TSP is to find the cheapest way of visiting all the cities precisely once and returning to the starting point.
If we look at the problem of bringing as many vehicles from one point to an-

other, we obtain the *maximum flow problem* in a network. In the *minimum cost flow problem* the aim is to bring a given number of vehicles as cheaply or as fast as possible to their destination.
A common and important question in the mentioned models is to find the *shortest path* from city A to city B over time. This can be done for a given starting time or a starting interval in which one wants to leave city A.
The outline of the thesis is as follows. In the next chapter we give some basic mathematical definitions, notation and concepts. In Chapter 3 we give a background of some well-known mathematical problems which we all extend with time dependent arcs in the remaining chapters. In Chapter 4 we start with the time dependent shortest path problem. An algorithm to solve this problem for a given starting time is given and proved. This proof could not be found in the literature. Chapter 5 adds the element time to network flows. The first part introduces flows over time and gives an algorithm for this problem. In the second part we add congestion to the model and describe a way to solve these problems. The time dependent traveling salesman problem is discussed in Chapter 6. We give an iterative heuristic to get a good tour which can be found in reasonable time. In the following chapter we give a comprehensive literature overview about the extension of the TDTSP the time dependent vehicle routing problem. We briefly note a way to get a good solution for this problem. The last chapter presents conclusions and directions for future work.

2. Preliminaries

In this chapter we give some basic mathematical definitions, notation and concepts.
A *graph* is a pair $G = (V, A)$ where V is a finite set and A is a set of ordered pairs of V. The elements of V are called the *vertices*, sometimes they are called nodes, cities or customers. The elements of A are called the *arcs* and are directed. We denote the arc from vertex i to vertex j by (i, j). If the arcs are not directed they are called *edges*. The cardinality of V is $|V| = n$ and of A it is $|A| = m$. The vertices k that can be reached from i by an arc (i, k) are called the *neighbors* of i. A *loop* is an arc (i, i). In our thesis we assume that the considered graphs do not contain loops, nor multiple edges between the same pair of vertices, which are *parallel* arcs. A *path* in a directed graph G is a sequence of distinct arcs, $P = (v_1 \to v_2 \to v_3 \cdots \to v_k)$.
On the arcs we can define a distance, a capacity or a travel time. The *distance* is the length of an arc. The *capacities* give the size or volume of the arcs. The *travel time* gives the duration of traveling along the arc during a time interval. We assume that all arc capacities, arc costs, arc lengths, travel times, supplies and demands are non-negative and integral. If they are rational we can multiply them with a suitable number to make them integral.
A very common property in graphs or networks is the *first in first out* (FIFO) property, also called the non-passing property. In a graph with constant travel times the FIFO-property guarantees that if a vehicle leaves a city i for a city j at a given time, any vehicle traveling at the same speed that leaves city i for city j at a later time will arrive later at city j.
The *triangle inequality* means that for any set of distinct vertices i, j, k it is faster (shorter) to travel directly from vertex i to vertex k than to travel from i to j, and then from j to k. The *Euclidean distance* is the straight line distance between two points i and j in the plane. If we use $\sum_j$, we take the sum over all the arcs from vertex i to all its neighbors j.
If we use $\Theta(n)$ it states for a theoretical measure of the execution of an algorithm, usually the time or memory needed, given the problem size n.
If we talk about a *flow* through a network we mean the amount of goods or vehicles that are going from a given start vertex to a given end vertex.
A subset B of A is called a *cut* if $B = \delta^{out}(S)$ for some $S \subseteq V$, where

$\delta^{out}(S) := \delta_A^{out}(S)$ =set of arcs of A leaving S. An s-t cut is a cut with $s \in S$ and $t \in \bar{S}$. The capacity $u[S, \bar{S}]$ of an s-t cut is the sum of the forward arcs in the cut. The capacity of a cut is an upper bound on the maximum amount of flow that can be sent from the vertices in S to the vertices in $\bar{S}$. A *minimum cut* is an s-t cut whose capacity is minimum along all s-t cuts. With this definition there follows a theorem; the maximum flow is equal to the minimal cut in a network. For the proof see Chapter 3.2.
The *time-dependent graph* is defined as $G_T = (V, A, W)$, with V a set of vertices, A a set of arcs and W a set of positive valued functions. The set of functions in W are *arc-duration functions* $w_{ij}(t)$ on each arc $(i, j) \in A$. The arc-duration function, $w_{ij}(t)$, specifies how much time it takes to travel from vertex i to vertex j, if one departs from i at time t. In the literature $w_{ij}(t)$ is often called an *edge-delay function*. The FIFO-property in G_T states that if departing earlier from vertex i one arrives earlier at vertex j, if one travels the same route. So it is not advantageous to delay a departure time.
In almost all cases we assume that there is no waiting time on an arc or in a vertex needed. We assume this, because we look at road networks and we cannot expect a car to wait in the middle of the road or at a crossing point.
The *Branch and cut* method is a method for solving integer linear programs, that is, linear programming problems where some or all the unknowns are restricted to integer values. The method is a combination of branch and bound and cutting plane methods. When an optimal solution is obtained, and this solution has a non-integer value for a variable that is supposed to be integer, a cutting plane algorithm is used to find further linear constraints which are satisfied by all feasible integer points but violated by the current fractional solution. This process is repeated until either an integer solution is found or until no more cutting planes are found. After that the branch and bound part of the algorithm is started. We say that a problem is *polynomial-time solvable*, if it can be solved by a polynomial-time algorithm. A *polynomial-time algorithm* is an algorithm that terminates after a number of steps bounded by a polynomial in the input size. NP-hard (nondeterministic polynomial-time-hard) is a class of problems that are at least as hard as the hardest problems in NP. These problems cannot be solvable in polynomial-time, unless $P = NP$. To understand these complexity classes read [Gare].
A list of abbreviations is given in Appendix A.

3. Static paths and flows: background and literature

In this chapter we give a background of some well-known mathematical problems that are related to traffic networks. These problems or algorithms do not take time into account. At first, the basic problem, finding a shortest path in a network will be explained.

3.1 Shortest paths

Given a directed graph $G = (V, A)$ with weights on the arc lengths, the shortest path problem is the problem of finding a path between two vertices $s, t \in V$ such that the sum of the weights of the arcs is minimized among all paths connecting s to t. The problem can be formulated in several ways: find the shortest paths from a source vertex to all the other vertices in the graph, or the other way around, from all the vertices in the graph to one vertex, or find the shortest paths between every pair of vertices. There exist several algorithms to solve these problems, like Dijkstra's algorithm [Dijk] or the Bellman-Ford algorithm [Bell][Ford]. The algorithmic approaches can be classified into two groups: *label setting* and *label correcting*. The approaches vary in how they update the distance labels and how they "converge" toward the shortest path distances. Label setting algorithms are applicable only to shortest path problems defined on acyclic networks with arbitrary arc lengths and to shortest path problems with nonnegative arc lengths. The label correcting algorithms are more general and apply to all classes of problems. Label setting algorithms have a much better worst-case complexity bound.
There are a lot of applications of the shortest path problem, for example telecommunication, logistic management and route planning in road networks.

3.2 Maximum flows in a network

A related problem is the maximum flow problem. One can find network flows everywhere in our daily life. Some examples are telephone networks, manufacturing and distribution networks, for example food distribution, computer networks, rail networks and the national highway systems. In all of these problems the goal is to move the maximum amount of flow from one point to another. In the study of *network flow problems* we learn more about such networks and try to optimize them.
In a network flow problem we consider a graph $G = (V, A)$. The graph G is a network with a source vertex s, the starting point, and a sink vertex t, the destination. Each arc in the graph has an assigned capacity, which gives the amount of flow that can flow through the arc. The aim is to find the maximum flow that can be sent from s to t without exceeding the given arc capacities.

3.2.1 The maximum flow problem

Given is a graph $G = (V, A)$ with a capacity u_{ij} for each arc $(i, j) \in A$.
Define f_{ij} as the flow through arc (i, j) and let $f(x)$ be the value of the total flow through the network, i.e. $f(x) = \sum_j f_{sj} - \sum_j f_{js}$.
The problem is formulated as follows:
Maximize $f(x)$
Subject to
$\sum_j f_{ij} - \sum_j f_{ji} = 0$ with $i \neq s, t$
$0 \leq f_{ij} \leq u_{ij}$ for each arc $(i, j) \in A$.
We consider the maximum flow problem subject to the following assumptions:

- The graph G is directed. If it is not, transform the undirected graph into a directed graph by replacing each undirected edge by two arcs, one in each direction.
- The network does not contain a directed path from vertex s to vertex t composed only of infinite capacity arcs. If this is the case the maximum flow value is unbounded.
- Whenever an arc (i, j) belongs to A, arc (j, i) also belongs to A. This means that there are no one way roads.

The concept of residual networks plays a central role in the development of all the maximum flow algorithms. Given a flow f, the residual capacity r_{ij} of any arc $(i, j) \in A$ is the maximum additional flow that can be sent from vertex i to vertex j using the arcs (i, j) and (j, i). Consequently $r_{ij} = u_{ij} - f_{ij} + f_{ji}$, the capacity of the arc (i, j) minus the flow from i to j

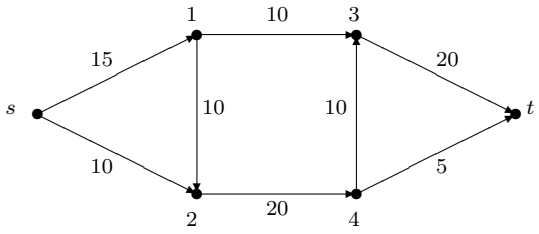
plus the flow from j to i. Call $G(f)$ the residual network, given flow f_x with x the current flow, consisting of the arcs with positive residual capacities. The advantage of the residual network $G(f)$ is that any path P from s to t in $G(f)$ gives a path, along which we can increase the flow. An augmenting path is a directed path from s to t in the residual network with positive residual capacity. The residual capacity of an augmenting path is the minimum residual capacity of any arc in the path.
In 1956 Ford and Fulkerson [Ford1] were one of the first who described an algorithm to solve this problem. Building the residual network and augmenting along an s-t path forms the core of the Ford-Fulkerson algorithm. They suggest the use of augmenting paths to change a given flow function in order to increase the total flow. If an arc on this path is directed from s to t, then push flow $f_{ij} \leq r_{ij}$ through it. (If an arc is in the opposite direction, then part of the flow can be returned.)

Algorithm Ford-Fulkerson

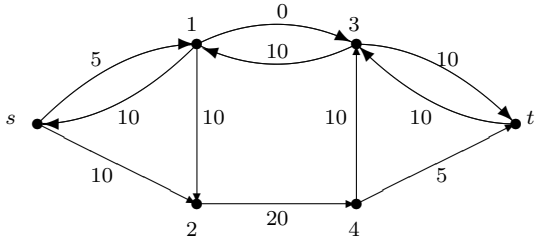
Step 0: Let $G(f) := G = (V, A)$ and start with $f_{ij} = 0,\ \forall (i,j) \in A$
Step 1: Search for augmenting paths
while there is an augmenting path P from s to t in $G(f)$ **do**
send a flow of value $f := \min_{i,j \in P} r_{ij}$ in $G(f)$ along P
augment this flow f in $G(f)$
construct a new residual network $G(f)$
end while
Step 2: If no augmenting path can be found, the algorithm terminates
return a maximal flow f with the amount of flow f_{ij} through arc (i,j).

In order to find an augmenting path for a given flow, a labeling process is used. At first vertex s is labeled. Then we label every vertex v, that can be reached by an augmenting path from s to v. After that vertex s is called *scanned*. If t is labeled, an augmenting path from s tot t has been found. This path is used to increase the total flow, and the procedure is repeated. Call I the set of vertices that is labeled, if a vertex is scanned, remove it from I. In the labeling process we call vertex i a *predecessor* of vertex j, if we first labeled vertex i and can reach vertex j from vertex i. In the first step of the following example we also give the labeling process.

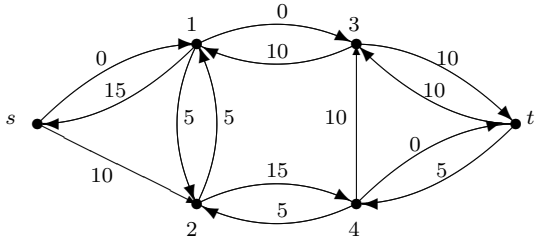
Example 1. *Ford-Fulkerson*
Given the following graph with capacity u_{ij} on the arcs, determine the total flow through the network.



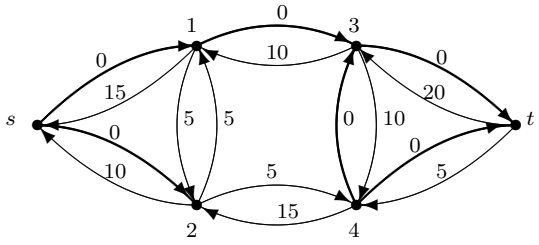
Iteration 1
1. $I = \{s\}$. *Vertex s is scanned and vertices 1,2 are labeled,* $pred(1) = pred(2) = s$.
2. $I = \{1, 2\}$. *Vertex 1 is scanned and vertex 3 is labeled,* $pred(3) = 1$.
3. $I = \{2, 3\}$. *Vertex 2 is scanned and vertex 4 is labeled,* $pred(4) = 2$.
4. $I = \{3, 4\}$. *Vertex 3 is scanned and vertex t is labeled,* $pred(t) = 3$.
Since vertex t is labeled there is a path from $s \to 1 \to 3 \to t$, *with* $min_{i,j \in P} r_{ij} = min\{15, 10, 20\} = 10$. *This gives the following flow:* $x_{s1} = x_{13} = x_{3t} = 10$. *We get the following residual graph:*



Iteration 2
There is a path from $s \to 1 \to 2 \to 4 \to t$ *with* $min_{i,j \in P} r_{ij} = min\{5, 10, 20, 5\} = 5$. *This gives the following flow:* $x_{s1} = 15$ $x_{12} = x_{24} = x_{4t} = 5$. *We get the following residual graph:*



Iteration 3
There is a path from $s \to 2 \to 4 \to 3 \to t$ *with* $min_{i,j \in P} r_{ij} = min\{10, 15, 10, 10\} = 10$. *This gives the following flow:* $x_{s2} = 10$ $x_{24} = 15$ $x_{43} = 10$ $x_{3t} = 15$. *We get the following residual graph:*



Iteration 4
There is no path from s tot t. There is a total flow of 25 through the network, with $x_{s1} = 15,\ x_{s2} = 15,\ x_{12} = 5,\ x_{13} = 10,\ x_{24} = 15,\ x_{43} = 10,\ x_{3t} = 20,\ x_{4t} = 5$.
In the last figure the thick lines give the cut.

We will now state the max-flow min-cut theorem, by Ford and Fulkerson [Ford1]:

Theorem 1. *The maximum value of the flow from a source vertex s to a sink vertex t in a capacitated network equals the minimum capacity cut among all s-t cuts*

Proof. If the optimal value of the flow is infinite, it is not hard to see that there must exist a directed path P from s to t, such that every arc in P has infinite capacity. For every cut S, there is an arc (i, j) belonging to path P such that $i \in S$ and $j \notin S$. Since that arc has infinite capacity, we conclude that $u[S, \bar{S}] = \infty$. Since this is true for every cut, we conclude that the minimal cut capacity is infinite and equal to the maximum flow value.
Suppose now that the optimal value, denoted by f^*, is finite. (This implies that there exists an optimal solution, that is, a flow whose value is f^*.) Let us apply the Ford-Fulkerson algorithm, starting with an optimal flow and the corresponding residual graph. Due to optimality of the initial flow, no flow augmentation is possible, and the algorithm terminates at the first iteration. Let S be the set of labeled vertices at termination. Since the search for an augmenting path starts by labeling s, we have $s \in S$. On the other hand, since no augmenting path was found, t is not labeled i.e. $t \in \bar{S}$ where $\bar{S}$ is the complement of S. Therefore the set $(S, \bar{S})$ is a cut. For every arc $(i, j) \in A$, with $i \in S$ and $j \notin S$ we must have $f_{ij} = u_{ij}$, otherwise vertices j would have been labeled by the labeling algorithm. Thus the total amount of flow that crosses the set $(S, \bar{S})$ is equal to $u[S, \bar{S}]$, so $u[S, \bar{S}] = f^*$. It follows that $u[S, \bar{S}]$ is the minimum cut capacity and it is equal to the value of the maximum flow. □

Since this theorem relates the optimal values of a minimization and a maximization problem, it is a duality theorem. More information can be found in [Bert].

3.2.2 The minimal cost flow problem

The *minimal cost flow problem* determines a least cost flow through the network that satisfies demands at the vertices from available supplies at vertices. Let $G = (V, A)$ be a directed graph where each arc $(i, j) \in A$ has a cost c_{ij} per unit flow on the arc. Each arc (i, j) also has a capacity u_{ij} that denotes the maximum amount of flow on the arc. Associated with each vertex $i \in V$ is an integer number $b(i)$ representing its supply, $b(i) > 0$, and its demand, $b(i) < 0$. The problem has a feasible solution when $\sum_{i=1}^{n} b(i) = 0$. If the (starting) problem does not satisfies this requirement dummy vertices are introduces.
The minimum cost flow problem is an optimization model formulated as follows:
Minimize $\sum_{(i,j)\in A} c_{ij}x_{ij}$
subject to $\sum_j x_{ij} - \sum_j x_{ji} = b(i)$ for all $i \in V$,
$0 \leq x_{ij} \leq u_{ij}$ for all $(i, j) \in A$.
For more information about network flows and its variations see the book *Network Flows* [Ahuj].

3.3 The traveling salesman problem

The problem of finding the shortest path that goes through every vertex of the graph exactly once, and returns to the start, is called the *traveling salesman problem* (TSP). The TSP is NP-hard. If the conjecture $P \neq NP$ is true, then a polynomial-time algorithm for solving the TSP does not exist, which implies that the worst-case running time for any algorithm for TSP increases exponentially with the number of vertices in the graph. Some instances with only hundreds of vertices could take many CPU (central processing unit) years to solve exactly. So, this is another kind of problem compared to the shortest path problem and maximum flow problem, which can be solved in polynomial-time.
Since the 1950's and 1960's the popularity of the problem has increased, because it plays a central role in logistics and also in algorithm development in combinatorial optimization. Danzig, Fulkerson and Johnson introduced in 1954 [Dan2] an integer linear programming formulation and solved the problem for 49 vertices (cities) with a cutting plane method. In the 1970's and 1980's it was possible to solve instances with up to 2392 vertices using cutting planes and branch and bound. The largest instance solved to optimality, has 85900 vertices (cities) and was solved in 2006 [Appl].
An often used heuristic to find good feasible solutions to the TSP is the nearest neighbor heuristic (NNH). The running time for this heuristic is $\mathcal{O}(n^2)$ and if the triangle inequality holds the worst-case solution value is $\Theta(\log n)$ times the optimal solution, with n being the number of vertices.

Christofides [Chri] developed an approximation algorithm that for any instance satisfying the triangle inequality produces a feasible solution with value less than or equal to $\frac{3}{2}$ times the optimal value. This is the best approximation algorithm known for this variant of the TSP. If the lengths of the arcs represent Euclidean distances then the best approximation algorithm is a $(1 + \frac{1}{\epsilon})$-algorithm with $\epsilon > 1$ [Aror].
In the asymmetric TSP, the distance between two vertices in the underlying network is not necessary the same in both directions. When the triangle inequality holds, the best approximation ratio known is obtained by Asadpour et al. and gives a solution within a factor $\mathcal{O}(\log n/\log \log n)$ of the optimum with high probability [Asad]. If no triangle inequality is imposed, there is no polynomial-time algorithm with constant approximation guarantee, for the asymmetric TSP. For more information about the TSP see [Appl2] or www.tsp.gatech.edu.

3.4 The vehicle routing problem

When the TSP is expanded with more than one route, it is called the *vehicle routing problem* (VRP). The VRP plays a vital role in distribution and logistics. We all make use of the system around us with routed messages, goods or people from one place to another. In the modern world we need vehicle routing to structure and find the optimal routes. The VRP is the problem of finding a set of shortest routes with a minimal cost for a fleet of vehicles. The set of vertices (costumers) is divided into subsets, such that each subset is serviced by one vehicle. Each vehicle starts and ends at the depot. In the basic problem it is assumed that each vehicle has the same fixed capacity. The sum of the demands of the visited customers on a route must not exceed the capacity of the vehicle. The problem has been analyzed extensively in the literature. Since the problem is NP-hard it is unlikely that a polynomial-time algorithm will be developed for determining its optimal solution. Consequently a great deal of work has been devoted to the development of heuristic algorithms.
Below we present several variations of the VRP.

- The VRP with time windows (VRPTW). Each customer must be visited within its own given time window. The depot also has its own time window. The time windows are defined as an interval in which the vehicle is allowed to arrive and depart at each customer (vertex). The costumers can have soft or hard time windows. When there are hard time windows there is a strict lower and upper bound. If the vehicle arrives before the lower bound, waiting time has to be taken into account. If it arrives too late the problem becomes infeasible. If there are soft time windows a vehicle can arrive before the lower bound or after the upper bound. If the vehicle is too early it has to wait, which

incurs cost. If it is too late it gets a penalty for lateness.

- There are a lot of different vehicles possible. The vehicle fleet can be heterogeneous, with each vehicle k having a different capacity b_k. It is also possible to have multiple capacity constraints. For example if there are both weight and volume restrictions.

- A vehicle may be capable of making more than one trip within a planning period, or a vehicle may both deliver and pick up products.

- There may be multiple depots with each vehicle assigned to a particular depot. Then the problem can be split into small VRPs with one depot.

- In the on-line VRP not all the customers are known at the start of the route. At every costumer, the vehicle has to calculate a new route.

3.4.1 Solution methods for the vehicle routing problem

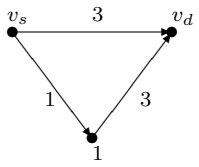
Huge research efforts have been devoted to studying the VRP since 1959 when Dantzig and Ramser [Dant] first described the problem mathematically. Many exact methods have been used to solve the VRP, such as algorithms based on linear programming techniques. Besides that, to find good feasible solutions for large-scale VRPs by heuristic techniques have received wide interests. The simple heuristics can be grouped into three categories: route building heuristics, route improvement heuristics, and two-phase methods.
Route buildings heuristics select arcs sequentially until a feasible solution has been created. For example start with a solution in which every city is supplied individually by a separate vehicle. This is a very expensive solution and if there are not enough vehicles it gives an infeasible solution. By combining any two of these routes, we would use one vehicle less and also reduce distance. Arcs are chosen based on some distance minimization criterion subject to the restriction that the selection does not create a violation of vehicle capacity constraints.
A route improvement heuristic begins with a set of arcs $S \subseteq A$ that constitutes a feasible schedule and seeks an interchange of a set $S_1 \subset S$ with a set $S_2 \subset A - S$ that reduces distance while maintaining feasibility.
Two-phase methods first assign cities to vehicles without specifying the sequence in which they are to be visited. Second, the routes are obtained for each vehicle using some TSP algorithm. An example is the 'Sweep' algorithm which is only applicable to problems in which cities are located at points in the plane and c_{ij} is the Euclidean distance. The cities are represented in a system with the origin at the depot. A city is chosen at random and the ray from the origin through the city is swept either clockwise or

counterclockwise. Cities are assigned to a given vehicle as they are swept, until further assignments of cities would exceed the capacity of that vehicle. Another class of heuristics are mathematical programming based heuristics, which are very different in character from the simple heuristics. This line of research began in the mid to late 70's when a number of researchers began to apply the machinery of mathematical programming to the VRP. For more information about the VRP see [Ball].
Recently, VRP exact algorithms have been based on either branch-and-cut or Lagrangian relaxation/column generation. Fukasawa et al. [Fuka] described in 2006 an algorithm that combines both approaches, which works over the intersection of two polytopes. The methode of Fukasawa et al. is equivalent to a linear program with exponentially many variables and constraints that can lead to lower bounds that are superior to those given by previous methods. Their branch-and-cut-and-price algorithm can solve, to optimality, all instances known from the literature with up to 135 vertices.

4. Time dependent shortest paths

Due to the increasing interest in dynamic management of transportation systems, it is important to find shortest paths where the weights (or delays) associated with arcs dynamically change over time. If real-time traffic information is available and the standard traffic patterns are known with high probability, it becomes possible to provide users with better services, such as "how to travel from one city to another city as fast as possible", by taking rush hour congestion into consideration.
To formulate this problem mathematically we need the definition of a time-dependent graph. A *time-dependent graph* is defined as $G_T(V, A, W)$ with V a set of vertices, A a set of arcs and W a set of positive valued functions. For every arc $(i, j) \in A$, there is an arc-duration function $w_{ij}(t) \in W$, where t is a time variable in a time domain $\mathcal{T}$. The function $w_{ij}(t)$ specifies how much time it takes to travel from vertex i to vertex j, if departing i at time t. There are different ways to define the arc duration function. It can be defined to be continuous, or stochastic, or to be the speed of an arc that depends on a time interval. From now on we call the source vertex v_s and the sink vertex v_d, because we need t to indicate time.

Example 2. *Time-dependent graphs*

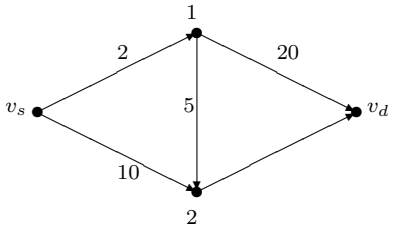


Given the graph above, with arc lengths like described in previous chapters. We see that the shortest path from v_s to v_d is the path $v_s \to v_d$. Now we introduce a time dependent arc $(1, v_d)$ and give the duration to travel through the arc if we leave from vertex 1 somewhere in the time interval $[0, 5]$. So the length of $(1, v_d)$ is no longer 3, it is 4 in the interval $[0, 2)$ and 1 in $[2, 5]$. If we start at $t = 1$ and take the path $v_s \to 1 \to v_d$, the length of our path

is 2. This is the fastest way to travel from vertex v_s to vertex v_d if we take time into account.

We assume that the time dependent graph satisfies the FIFO-property on the arcs. The FIFO-property in G_T states that if departing earlier from vertex i one arrives earlier at vertex j, if one travels the same route. So it is not advantageous to delay a departure time. For the time dependent arcs this mean that every arc (i, j) has the FIFO-property, if
$\forall\ t_\Delta \geq 0 : w_{ij}(t_i) \leq t_\Delta + w_{ij}(t_i + t_\Delta)$
or $t_i + w_{ij}(t_i) \leq t_j + w_{ij}(t_j)$ for $t_i \leq t_j$
Where t_i is the departure time at vertex i and t_Δ a small time interval.
It is also possible to transform a non-FIFO time dependent graph into a time dependent graph with FIFO-property on the arcs by inserting some waiting time on each vertex in the optimal shortest path.

Example 3. *From non-FIFO to FIFO*



Suppose that the durations on arcs $(v_s, 1)$, $(v_s, 2)$, $(1, 2)$ and $(1, v_d)$ are constant over time, and are given in the graph above. The duration on arc $(2, v_d)$ is equal to 10 in interval $[0, 9]$ and and equal to 5 in interval $(9, 20]$. Assume we start at $t = 0$. At $t = 2$ we arrive at vertex 1. We leave vertex 1 at $t = 2$ and arrive at vertex 2 at $t = 7$. This looks like the fastest path to follow. When we leave vertex 2 at time $t = 7$ we arrive at vertex v_d at $t = 17$.
If we left vertex 2 at $t = 10$ (so we came from arc $(v_s, 2)$) we would arrived at vertex v_d at $t = 15$. This means that if we leave vertex 2 later we will arrive earlier on vertex 2. This is a non-FIFO graph.
To make the FIFO-property hold, we introduce a waiting time. In this example you can see that if you travel through vertex 1 (which is the quickest way to go to vertex 2) it is faster path to wait a time period of 3 at vertex 2.

For all the functions $w'_{ij}(t)$ in the non-FIFO graph G'_T we define $w_{ij}(t)$ to construct a FIFO graph G_T.
$w_{ij}(t) = \Delta_{ij}(t) + w'_{ij}(t + \Delta_{ij}(t)) = \min_{0 \leq t_\Delta \leq t_d - t}\{t_\Delta + w'_{ij}(t + t_\Delta)\}$
Here t_t is the end time, $\Delta_{ij}(t)$ is the minimal waiting time required at vertex i to go to vertex j, to make the graph satisfy FIFO and t_Δ all the possible time intervals to wait. Since the starting time interval T is a closed interval, $w_{ij}(t)$ and $\Delta_{ij}(t)$ are well defined. If there are multiple possible values of t_Δ to minimize $t_\Delta + w'_{ij}(t + t_\Delta)$, we select any of them as $\Delta_{ij}(t)$. It also satisfies to keep the slope of the arc duration functions between 1 and -1.

4.1 A shortest path algorithm for a given starting time

Given a source v_s and a starting time t_s, we look for the shortest paths and minimum delays between v_s and all other vertices for that starting time. In 1969 Dreyfus [Drey] already noted that there exist straightforward extensions to such algorithms as Dijkstra [Dijk] or Ford [Ford3]. We will give the algorithm described in Orda and Rom [Orda], which is a so-called labeling algorithm. Each vertex is labeled by the earliest possible arrival time at that vertex, calculated with the given starting time at the source. The vertices get temporary labels Y_k and permanent labels X_k ($NULL$ indicating the vertex is not permanently labeled). A temporal label gives the length of the shortest path to vertex k, that we found until then. A permanent label gives the length of the shortest path to get to that vertex. The algorithm also saves the predecessor $pred(k)$ for all permanent labeled vertices k, so the vertices that are visited on the shortest path can be found effectively. The input is a time dependent graph with functions $w_{ij}(t)$ and a given starting time t_s.

Algorithm Time dependent shortest path with a given start time

Step 0: Initialization
$X_s = t_s;\ pred(s) = 0;\ \forall k \neq s\ Y_k = \infty,\ X_k = NULL,\ pred(k) = 0;\ j = s;$
Step 1:
for all $k : (j,k) \in A$ for which $X_k = NULL$ **do**
a:
$Y_k = \min\{Y_k, X_j + w_{jk}(X_j)\}$
b:
if Y_k changed in **Step 1a then**
Set $pred(k) = j$
end if
end for
Step 2:
if all vertices have no null X-value **then**
Stop
else if l is a vertex for which $X_l = NULL$ and such that $Y_l \leq Y_k\ \forall k$ for which $X_k = NULL$ **then**
Set $X_l = Y_l,\ j = l$ and proceed with **Step 1**.
end if

The main difference between a conventional shortest path algorithm and the algorithm above is the calculation of $w_{jk}(X_j)$ in step 1a. In this step we take the path with the shortest length in time. Because the starting time and the functions $w_{ij}(t)$ are given, we can calculate $w_{jk}(X_j)$ for every X_j. Then it is just the length of the arc, and with a proof similar to Dijkstra's

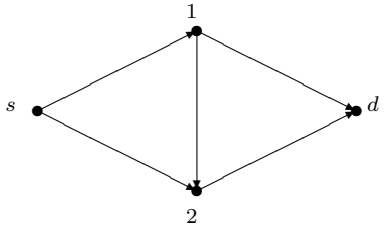
algorithm the correctness is proven. The algorithm terminates after $\mathcal{O}(|V|^2)$ operations. Each path from v_s to any vertex j is a shortest path for starting time t_s whose duration is given by $X_j - t_s$. In the literature we could not find a correctness proof of this algorithm so we give our own proof, inspired by the proof of Dijkstra's algorithm.

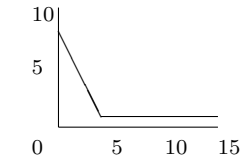
Theorem 2. *The algorithm terminates after $\mathcal{O}(|V^2|)$ operations. After execution, each path from v_s to any vertex j is a shortest path from starting time t_s whose delay is given by $X_j - t_s$.*

Proof. Let u be the vertex which gave v its present label Y_v; namely, $X_u + w_{uv}(t_u) = Y_v$, with $w_{uv}(t_u)$ a fixed length (duration) of the path from u to v. After this assignment took place, u did not change its label, since we have chosen u because it has a permanent label. It is not possible that if departing later from vertex u one arrives earlier at vertex v, if one travels the same route, because of the FIFO-property. Next, find the vertex which gave u its final label Y_u with $pred(u)$, and repeating this backward search, we trace a path from v_s to v whose length is exactly Y_v. The backward search finds, in every step, a vertex that has obtained a permanent label at a previous step, and therefore no vertex on this path can occur more than once; it can only terminate in v_s, which has been assigned its label in Step 0.
Let us look at the complexity of the algorithm. Let $Y \subseteq V$ be the set of vertices with a temporary label. In Step 1a one considers each arc connected to vertex j exactly once. Thus it uses, at most, $\mathcal{O}(|A|)$ time. Since we have no loops or parallel arcs, $|A| \leq |V| \times (|V| - 1) \leq |V|^2$. Step 1b is of $\mathcal{O}(1)$ and is repeated $|V|$ times. In Step 2 the minimum label of the elements of Y has to be found. At the start of the algorithm Y consists of all vertices, $Y = V$, and then the cardinality of Y decreases by one each time. This can be done in $|Y| - 1$ comparisons and the search is repeated $|V|$ times. Thus the total time spent in Step 2 is $\mathcal{O}(|V|^2)$. Thus, the whole algorithm is of $\mathcal{O}(|V|^2)$ complexity. □

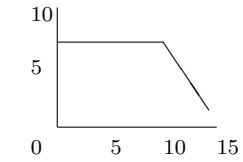
Example 4. *Shortest path for a given starting time*

Given the following graph with the arc duration functions that are shown in the graphics below.

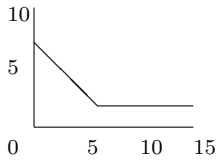




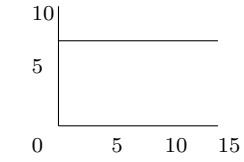
$w_{s1}(t)$



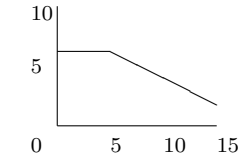
$w_{s2}(t)$



$w_{12}(t)$



$w_{1d}(t)$



$w_{2d}(t)$

To find the shortest path in time for start time 2 we use the algorithm above.
Iteration 1.
0. $X_s = t_s = 2$ $pred(s) = 0$ $\forall k \neq s$ $Y_k = \infty$ $X_k = NULL$ $pred(k) = 0$ $j = s$
1a. $k = \{1, 2\}$
$Y_1 = min\{\infty, X_s + w_{s1}(X_s)\} = min\{\infty, 2 + w_{s1}(2)\} = min\{\infty, 2 + 5\} = 7$
$Y_2 = min\{\infty, X_s + w_{s2}(X_s)\} = min\{\infty, 2 + w_{s2}(2)\} = min\{\infty, 2 + 8\} = 10$
1b. $pred(1) = s$ $pred(2) = s$
2. $l = 1$ $X_1 = 7$ $j = 1$
Iteration 2.
1a. $k = \{t, 2\}$
$Y_d = min\{\infty, X_1 + w_{1d}(X_1)\} = min\{\infty, 7 + w_{1d}(7)\} = min\{\infty, 7 + 8\} = 15$
$Y_2 = min\{10, X_1 + w_{12}(X_1)\} = min\{10, 7 + w_{12}(7)\} = min\{10, 7 + 2\} = 9$
1b. $pred(d) = 1$ $pred(2) = 1$
2. $l = 2$ $X_2 = 9$ $j = 2$
Iteration 3.
1a. $k = \{d\}$
$Y_d = min\{15, X_2 + w_{2d}(X_1)\} = min\{15, 9 + w_{2d}(9)\} = min\{15, 9 + 5\} = 14$
1b. $pred(t) = 2$
2. $l = t$ $X_t = 14$ $j = d$
So we found a shortest path $s \to 1 \to 2 \to d$ *of length 14.*

4.2 A shortest path algorithm for a given starting interval

Recently Ding, Xu Yu, and Qin (2009)[Ding] described an algorithm to find time-dependent shortest paths (TDSP) over large graphs. They studied how to answer queries of finding the best departure time that minimizes the total travel time from one place to another, over a road network, where the traffic conditions dynamically change over time. The problem is to find the

optimal path with the minimal travel time from a source v_s to a destination v_d, over a time-dependent graph, when the starting time can be selected from a user-given starting time interval T. In the time-dependent graph the functions $w_{ij}(t)$ are piece-wise linear functions on the arcs (see Example 4). The function $g_i(t)$ represents the v_s to v_i earliest arrival time function. They use a user specified query, where the user is asked to select the minimum travel time path, from v_s to v_d, with the best departure time from a time interval T. They consider FIFO time-dependent graphs, as well as general time-dependent graphs. First the authors focus on finding an answer in a FIFO time dependent directed graph G_T, where no waiting time is needed in optimal solutions. They propose a Dijkstra based algorithm to find the optimal solution which consists of two steps.
First it uses the algorithm *TimeRefinement* that computes for every vertex $v_i \in V$ the earliest arrival time function $\{g_i(t)|v_i \in V\}$, when departing from v_s at any starting time $t \in T$. The input is a time dependent graph G_T, a user specified query and starting time interval $T = [t_s, t_d]$.

TimeRefinement($G_T(V, A, W),\ v_s,\ v_d,\ T$)
  $g_s(t) = t$ for $t \in T;\ \tau_s = t_s$;
  for each $v_i \neq v_s$ **do**
    $g_i(t) = \infty$ for $t \in T;\ \tau_i = t_s$;
  end for
  Let Q be a priority queue initially containing pairs $(\tau_i, g_i(t))$, for all vertices $v_i \in V$, ordered by $g_i(\tau_i)$ in ascending order;
  while $|Q| \geq 2$ **do**
    $(\tau_i, g_i(t)) = dequeue(Q)$;
    $(\tau_k, g_k(t)) = head(Q)$;
    $update(Q, (\tau_j, g_j(t)))$;
  end while
  $\tau_i = \tau_i'$;
  if $\tau_i \geq t_d$ **then**
    if $v_i = v_d$ **then**
      return $\{g_i(t)|v_i \in V\}$
    else
      $enqueue(Q, (\tau_i, g_i(t)))$;
    end if
  end if
  return $\{g_i(t)|v_i \in V\}$.

Here Q is the query. The operation $dequeue(Q)$ dequeues the top pair from Q, $head(Q)$ retrieves a new top pair in the queue and at $enqueue(Q)$ the pair that was dequeued is placed back if it has not been well refined in the entire starting time interval.
In the second step the algorithm *PathSelection* selects one of the paths

from v_s to v_d that matches the optimal travel time. The input is the time-dependent graph G_T, the set of earliest arrival-time functions $g_i(t)$, computed from *TimeRefinement*, for all vertices $v_i \in V$, source v_s, destination v_d and the optimal starting time t^*.

PathSelection($G_T(V, A, W), \{g_i(t)\},\ v_s,\ v_d,\ t^*$)

```
v_j = v_d;
p* = ∅;
while v_j ≠ v_s do
  for each (v_i, v_j) ∈ A do
    if g_i(t*) + w_{i,j}(g_i(t*)) = g_j(t*) then
      v_j = v_i
    end if
  end for
  p* = (v_i, v_j) × p*;
end while
return p*;
```

For a technical description of the algorithms, an example and the correctness proofs, see the article by Ding et al. It is shown that the time complexity is $\mathcal{O}((n\log n + m)\alpha(T))$, where $\alpha(T)$ is the cost required for each function operation. This $\alpha(T)$ is possibly unbounded because it depends on the relative values of arrival time functions. For an unbounded instance see [Dehn].

4.2.1 Time functions with time intervals

Sung et al. [Sung] presented a flow speed model where the flow speed of each arc depends on the time intervals. To satisfy the FIFO-property the flow speed of each arc and not the travel time changes as the interval changes. They use a Dijkstra label setting shortest path based algorithm.
Consider a network $G = (V, A)$ and let l_{ij} be the non-negative length of the arc (i, j). Divide the time horizon into the following time intervals; $[f_k, f_{k+1}), k = 0, 1, 2, \ldots, K-1$. Let $v_{k_{(ij)}}$ be the non-negative flow speed in time interval $[f_k, f_{k+1})$ on arc (i, j). Define a value $t_i = \min_{j \neq i}\{T(t_i, (i, j))\}$ where $T(t_i, (i, j))$ is the arrival time at vertex j starting from vertex i at time t_i. In the algorithm, the travel time of each arc is calculated according to the flow speed at the time of passing the arc. They only insert a function calculating the arrival time at the next connecting vertex of the current vertex in the Dijkstra algorithm.
Let $pred(i)$ be the predecessor vertex of i and let $A(i)$ be the set of all arcs connected to i. The set of vertices which are not scanned is given by U and S is the set of scanned vertices. Then the combined Dijkstra's label setting algorithm is described as follows:

Algorithm modified Dijkstra

$S := \emptyset;\ U := V$
$t_i = \infty$ for each vertex $i \in V$
$t_s = 0$ and $pred(s) = 0$
while $|S| < n$ **do**
  let $i \in U$ be a vertex for which $t_i = \min\{t_j : j \in U\}$
  $S := S \cup \{i\}$
  $U := U - \{i\}$
  for each$(i, j) \in A(i)$ **do**
    if $t_j > Arrivaltime(t_i,\ (i, j))$ **then**
      $t_j := Arrivaltime(t_i,\ (i, j))$ and $pred(j) := i$
    end if
  end for
end while

Let $Arrivaltime(t_i,\ (i, j))$ be the arrival time at vertex j starting from vertex i at time t_i. The function that calculates the arrival time is the following function:

Arrivaltime$(t_i,\ (i, j))$

$Arclength := l_{ij}$
let $k \in \{0, 1, 2, \ldots, K\}$ be an index for which $f_{k_{(i,j)}} \leq t_i < f_{k+1_{(i,j)}}$
$Arclength := Arclength - v_{k_{(i,j)}} \times (f_{k+1_{(i,j)}} - t_i)$
while $Arclength > 0$ **do**
  $k := k + 1$
  $Arclength := Arclength - v_{k_{(i,j)}} \times (f_{k+1_{(i,j)}} - f_{k_{(i,j)}})$
end while
$Arrivaltime(t_i,\ (i, j)) := f_{k+1_{(i,j)}} + Arclength/v_{k_{(i,j)}}$
return

The computational complexity of the original Dijkstra's algorithm is $\mathcal{O}(n^2 + m)$, where n are the number of vertices in the network en m the number of arcs. For the modified algorithm it gives $\mathcal{O}(n^2 + mK)$, where K is the maximum number of time intervals scanned in *Arrivaltime*.

5. Time dependent flows

5.1 Network flows over time

Flow variation over time is an important feature in network flow problems arising in various applications, an example is road traffic control. Traffic flows have two important features that make them difficult, namely congestion and time. Congestion captures the fact that travel times increase with the amount of flow on the roads and time refers to the movement of cars along a path as a flow over time. These aspects are not captured by the classic static networks.
Network flows over time include a temporal dimension and therefore provide a more realistic modeling tool for numerous real world applications. We will concentrate on flows over time (also called "dynamic flows" in the literature) with finite time horizon and constant capacities and constant transit times in a continuous or discrete time model. Here the flow requires a certain amount of time to travel through each arc. So not only the amount of flow is taken into account, also the time it takes to travel through the network.

5.2 Ford and Fulkerson

Ford and Fulkerson first introduced the notion of flows over time and constructed in 1958 [Ford2] an algorithm to find a maximal dynamic flow from a static flow. The problem is to find the maximal amount of flow that can be transported from one vertex to another in a given number T of time periods, and to determine along which arcs the flow is sent in order to achieve this maximum. In the article by Ford and Fulkerson there is a computationally efficient algorithm for solving this dynamic linear programming problem presented. They also give an extensive example on how to use their algorithm. We will give the problem definition and their algorithm. Let $G = (V, A)$ be a directed graph with a source vertex $v_s \in V$ and a sink vertex $v_d \in V$. Each arc $(i, j) \in A$ has a capacity u_{ij} and a transit time t_{ij}. If the graph includes costs, each arc also has a cost coefficient. The total number of time periods, is specified by T. Let $v(T)$ denote the total amount of flow that leaves the source and enters the sink during the T periods. Let $\delta^+(v)$ and

$\delta^-(v)$ denote respectively, the set of arcs $(i,j) \in A$ leaving vertex v and entering vertex v.
A (static) network flow x assigns a non-negative flow value x_{ij} to each arc $(i,j) \in A$. The flow is feasible if it respects the capacity constraints: $0 \leq x_{ij} \leq u_{ij}$. An s-d flow x satisfies flow conservation at each vertex $v \in V \backslash \{v_s, v_d\}$, that is, $\sum_{(i,j)\in\delta^-(v)} x_{ij} - \sum_{(i,j)\in\delta^+(v)} x_{ij} = 0$.

5.2.1 The algorithm from Ford and Fulkerson

The algorithm from Ford and Fulkerson [Ford2] constructs a static flow and is essentially a primal dual method for a capacitated transshipment problem. The algorithm is an iterative process that has as final output an integral static flow x_{ij}, together with a set of integers $\pi(i)$ for each vertex i in V. $\pi(i)$ is the so-called vertex potential and gives the travel time, with respect to transit times of arcs, from v_s to vertex i. After augmenting flow along the shortest s-d path P^k in G_{x^k}, $\pi^k(i)$ is still feasible and k gives the iteration number. Here G_{x^k} is the residual network with flow x^k. This implies that the shortest path distances have not been decreased. All the $\pi(i)$ have to satisfy the following constraints:

$$\pi(s) = 0,\ \pi(t) = T + 1,\ \pi(i) \geq 0 \tag{5.1}$$

$$\pi(i) + t_{ij} > \pi(j) \text{ for } x_{ij} = 0 \tag{5.2}$$

$$\pi(i) + t_{ij} < \pi(j) \text{ for } x_{ij} = u_{ij} \tag{5.3}$$

To start the algorithm take all $x_{ij} = 0$ and all $\pi(i) = 0$.
Arcs (i,j) for which $\pi(i) + t_{ij} = \pi(j)$ will be called admissible arcs. Note that at most one member of the pair $(i,j), (j,i)$ will be admissible. During the algorithm a vertex is either unlabeled, or labeled but unscanned, or labeled and scanned. If a vertex gets a label like $[v_k^{\pm}, h]$, the label says that vertex v_k is the predecessor of this vertex and it is possible to send a flow with value h to this vertex. The position of the $\pm$ gives the direction of the flow, a $+$ if it is directed from v_s to v_d and $-$ if the flow is directed in the opposite direction. A vertex is scanned if all the labels of its neighbors are updated. Start with all vertices unlabeled.

Algorithm Ford and Fulkerson with time

Step 1: Give v_s the label $[v_d^+, \infty]$, and consider v_s as unscanned.

Step 2: Select any labeled, unscanned vertex v_i, suppose it is labeled $[v_k^{\pm}, h]$.

a: To all vertices v_j that are unlabeled and such that (i,j) is admissible and $x_{ij} < u_{ij}$ assign the label $[v_i^+, \min(h, u_{ij} - x_{ij})]$. All v_j are now labeled and unscanned.

b: To all vertices v_j that are now unlabeled and such that (j,i) is admissible and $x_{ji} > 0$ assign the label $[v_i^-, \min(h, x_{ji})]$. Such v_j are now labeled and unscanned.

c: Now define v_i to be labeled and scanned.

Repeat this step until the sink v_d is labeled and unscanned, or until no more labels can be assigned. In the former case go to Step 3. In the latter case, let the present flow be denoted by the new name $x_{ij}^{'}$, and go to Step 4.

Step 3: Update the flow.

a: If v_d is labeled $[v_k^+, h]$ replace x_{kd} by $x_{kd} + h$.

b: If v_d is labeled $[v_k^-, h]$ replace x_{kd} by $x_{kd} - h$.

c: Go to vertex v_k and treat it the same way as vertex v_d.

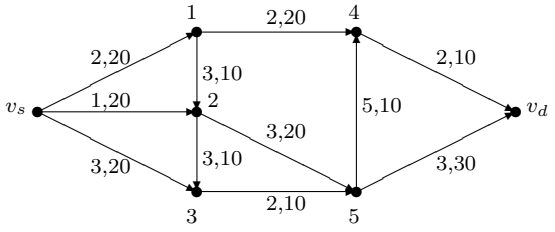
Repeat until v_s is reached. Now the replacement process is stopped. This process alters the flow along a path from v_d to v_s.
Discard all labels and return to Step 2 with the new flow.

Step 4: Define $\pi^{'}(i)$ by

$$\pi^{'}(i) = \begin{cases} \pi(i) & \text{if } v_i \text{ is labeled;} \\ \pi(i)+1 & \text{if } v_i \text{ is unlabeled.} \end{cases}$$

Repeat the algorithm, starting with $x_{ij}^{'}$ and $\pi^{'}(i)$ and continuing until the value of $\pi(t)$ has been increased to $T+1$.

Example 5. *Ford and Fulkerson with time*
Given the following graph with capacity u_{ij} and time t_{ij} on the arcs, determine the total flow through the network.



Take for the time period $T = [0, 8]$ and start with $\pi(i) = 0$.
Iteration 1
a Label $v_s : [v_d^+, \infty]$
b There are no admissible arcs. v_d has no label.
d $\pi(v_s) = 0,\ \pi(i) = 1$ for $i \neq v_s$

Iteration 2

a Label $v_s : [v_d^+, \infty]$
b Label $v_2 : [v_s^+, 20]$. v_d *has no label.*
d $\pi(v_s) = 0,\ \pi(2) = 1,\ \pi(i) = 2$ *for* $i \neq v_s, 2$

Iteration 3
a Label $v_s : [v_d^+, \infty]$
b Label $v_1 : [v_s^+, 20]$. v_d *has no label.*
d $\pi(v_s) = 0,\ \pi(2) = 1,\ \pi(1) = 2,\ \pi(i) = 3$ *for* $i \neq v_s, 1, 2$

Iteration 4
a Label $v_s : [v_d^+, \infty]$
b Label $v_3 : [v_s^+, 20]$. v_d *has no label.*
d $\pi(v_s) = 0,\ \pi(2) = 1,\ \pi(1) = 2,\ \pi(3) = 3,\ \pi(i) = 4$ *for* $i \neq v_s, 1, 2, 3$

Iteration 5
a Label $v_s : [v_d^+, \infty]$
b Label $v_4 : [v_1^+, 20]$ *and* $v_5 : [v_2^+, 20]$. v_d *has no label.*
d $\pi(v_s) = 0,\ \pi(2) = 1,\ \pi(1) = 2,\ \pi(3) = 3,\ \pi(4) = \pi(5) = 4,\ \pi(v_d) = 5$

Iteration 6
a Label $v_s : [v_d^+, \infty]$
b No new labels can be added. v_d *has no label.*
d $\pi(v_s) = 0,\ \pi(2) = 1,\ \pi(1) = 2,\ \pi(3) = 3,\ \pi(4) = \pi(5) = 4,\ \pi(v_d) = 6$

Iteration 7
a Label $v_s : [v_d^+, \infty]$
b Label $v_d : [v_4^+, 10]$. v_d *has a label*
c There is a path from vertex v_s *to* v_d. *This gives* $x_{4d} = x_{14} = x_{s1} = 10$. *The arc* $(4, v_d)$ *is the cut. We delete all the labels.*
a Label $v_s : [v_d^+, \infty]$
b Label $v_2 : [v_s^+, 20],\ v_1 : [v_s^+, 10],\ v_3 : [v_s^+, 20],\ v_4 : [v_1^+, 10]$ *and* $v_5 : [v_2^+, 20]$. v_d *has no label.*
d $\pi(v_s) = 0,\ \pi(2) = 1,\ \pi(1) = 2,\ \pi(3) = 3,\ \pi(4) = \pi(5) = 4,\ \pi(v_d) = 7$

Iteration 8
a Label $v_s : [v_d^+, \infty]$
b Label $v_d : [v_5^+, 20]$. v_d *has a label*
c There is a path from vertex v_s *to* v_d. *This gives* $x_{5d} = x_{25} = x_{s2} = 20$. *There are no arcs used from the first path so we can send another flow through that path which starts 1 time period later. The arcs* $(4, v_d)$, $(v_s, 2)$ *and* $(2, 5)$ *form a cut. We delete all the labels.*
a Label $v_s : [v_d^+, \infty]$
b Label $v_1 : [v_s^+, 10],\ v_3 : [v_s^+, 20]$ *and* $v_4 : [v_1^+, 10]$. v_d *has no label.*
d $\pi(v_s) = 0,\ \pi(2) = 2,\ \pi(1) = 2,\ \pi(3) = 3,\ \pi(4) = 4, \pi(5) = 5,\ \pi(v_d) = 8$

Iteration 9
a Label $v_s : [v_d^+, \infty]$
b Label $v_2 : [v_1^+, 10]$, $v_5 : [v_3^+, 10]$ *and* $v_d : [v_5^+, 10]$. v_d *has a label*
c There is a path from vertex v_s *to* v_d. *This gives* $x_{5d} = 30$, $x_{35} = 10$, $x_{s3} = 10$. *There are no arcs used from the first path so we can send another flow through that path which starts 1 time period later. It is also possible to send a flow through the path* $(v_s, 2, 5, v_d)$ *one period later. The arcs* $(4, v_d)$, $(v_s, 2)$, $(2, 5)$, $(3, 5)$ *and* $(5, v_d)$ *form a cut. We delete all the labels.*
a Label $v_s : [v_d^+, \infty]$
b Label $v_1 : [v_s^+, 10]$, $v_2 : [v_1^+, 10]$, $v_3 : [v_s^+, 10]$ *and* $v_4 : [v_1^+, 10]$. v_d *has no label.*
d $\pi(v_s) = 0$, $\pi(2) = 2$, $\pi(1) = 2$, $\pi(3) = 3$, $\pi(4) = 4, \pi(5) = 6$, $\pi(v_d) = 9$
$\pi(v_d) = 9 = T + 1$ *so stop.*
We found the following flow:
- Path $(v_s, 1, 4, v_d)$ *with flow 10 which can be sent three times in the time interval, so the total flow through this path is 30.*
- Path $(v_s, 2, 5, v_d)$ *with a flow value of 20 and the flow can be sent two times in the time interval, so the total flow through this path is 40.*
- Path $(v_s, 3, 5, v_d)$ *with a flow value of 10 and this path can be sent one time in the time period.*
This gives a total flow in $T = [0, 8]$ *of 80.*

For more information about network flows over time M. Skutella wrote an introduction to network flows over time (2008) [Skut].

5.3 Flows over time with flow dependent transit times

In the model from Ford and Fulkerson there are discrete time steps used. The arcs have constant time durations, so they do not take congestion into account. If we look at networks with congestion without the transit times and capacity restrictions but full information about the traffic situation, users will try to choose a fastest route and achieve a so-called user equilibrium. In this equilibrium no driver can get a faster route through the network if everybody else stays with their current route. So it has a certain fairness, since all users of the same origin-destination pair have the same travel time. Therefore, it has been used as a reasonable solution to the static traffic problem with congestion. This equilibrium is often not the best solution to get everybody at their destination as fast as possible. The system optimum provides the best possible solution but it may have longer routes for some users and thus misses fairness. A way to find a solution that lies inbetween these solutions, is to use only roads closely around the road with the most congestion. Fore more information about this problem see

Köhler, Möhring and Skutella's article about traffic networks and flows over time [Kohl].
Now we will look at networks where the duration of an arc changes over time. The field of flows over time with flow dependent transit times has attracted attention only in the last couple of years. One reason for this seems to be the lack of well defined models for this kind of flows, which is due to the more complicated setting. There are hardly any algorithmic techniques known which are capable of providing reasonable solutions even for networks of small size.
One problem is how to define the transit time functions. Often they rely on relatively simple functions which are simplifications of the actual flow. Another problem is the definition of the transit time and flow on a particular arc. Do we take the flow at the beginning of the arc, at the moment that the flow enters the arc or is the total amount of flow that is on the arc considered? With congestion these two values could be different. Again, we take the FIFO-property into account, overtaking of flow units is not permitted.
A model that shows that there is at least a good temporally repeated flow for this problem was suggested by Köhler and Skutella [Koh2]. It is inspired by the earlier mentioned result of Ford an Fulkerson.
Another way to model this problem is based on the *time-expanded network*. This network contains one copy of the subgraph based on the vertices that can be reached in each discrete time step. So there are time layers created. A discrete time flow over time in the given network can be interpreted as a static flow in the corresponding time expanded network. One can apply the algorithms developed for static networks flows to this generalized time-expanded network. The underlying assumption for this approach is that at any moment of time the transit time on an arc only depends on the current rate of inflow into that arc. However, when considering large networks this model is not applicable because the time-expanded network will grow enormously.

6. Time dependent traveling salesman problem

The time dependent traveling salesman problem (TDTSP) is a generalization of the TSP where the travel time (cost) between two cities depends on the moment of the day the arc is traveled.
According to Picard and Queyranne (1978) [Pica] the TDTSP was introduced by Fox (1973), where it was illustrated with examples from the brewing industry. Several linear integer programming formulations were presented in that work but none was found to lead to a tractable solution scheme. The paper of Picard and Queyranne (1978) can be considered as one of the basic works about the TDTSP. It presents the first practically effective method for solving TDTSPs with up to 20 vertices. This method is based on shortest path computations, dynamic programming and a branch and bound algorithm. The travel time between two vertices depends on the position of the vertices in the sequence of the tour. Their model is still one of the few exact models proposed in the literature.
Malandraki and Daskin (1992) [Mala] described the problem as a special case of the TDVRP, where they treated the time function as a step function. In the next chapter we describe their approach. In 1996 Malandraki and Dial [Mal2] proposed a restricted dynamic programming heuristic (a generalization of the NNH). The heuristic lets the user choose some middle ground between the optimal dynamic programming algorithm and the NNH. This means that only a user-specified number of partial tours is allowed at each stage. Their heuristic gives a good solution for problems of more than 200 vertices. Vander Wiel and Sahinidis [Vand] developed a branch and cut algorithm and applied a Benders decomposition to the problem in order to speed up lower bound calculation and solved the TDTSP. They solved instances having up to 18 nodes to optimality, by embedding their bound in a branch-and-bound algorithm.
In 2008 Bigras et al. [Bigr] also presented an exact solution method for the TDTSP. They show how integer programming formulations of the TDTSP can be extended to single machine scheduling problems with sequence dependent setup times. Several integer programming formulations of varying

size and strength are introduced. There is an exact branch and bound algorithm proposed that is able to solve instances up to 45 vertices.
A more recent research by Miranda-Bront et al. (2010) [Mira] considers the models presented in Picard and Queyranne and Vander Wiel and Sahinidis. Both polytopes are analyzed and families of valid inequalities for both models are derived. They present computational results for a branch and cut algorithm that uses these inequalities. The results show that their algorithm seems very efficient compared to known algorithms.

6.1 A heuristic algorithm for TDTSP

Many companies deliver products daily and/or weekly to their costumers. To visit all their customers in an optimal way taking rush hours into account, we advice to use the branch and bound algorithm from Bigras et al. If there are about 25 customers or more it would be faster to use a heuristic, but then no guarantee of optimality can be given. We give a nearest neighbor heuristic that could be used for a problem with time-dependent travel times. Given is a time-dependent graph $G_T = (V, A, W)$, with $V = \{s, 1, 2 \ldots, n\}$ the set of vertices (customers), where $v_s = s, n$ are the depot. The set of piece-wise continuous functions in W are arc duration functions $w_{ij}(t)$ on each arc $(i, j) \in A$. The vertices have to be visited within a given time horizon T and t_s is the starting time for the tour.
In the heuristic we present below, the last vertex of the (sub)route is defined by l, Q is the set of unvisited vertices and $\mathcal{R}$ is an ordered set and gives the (sub)route.

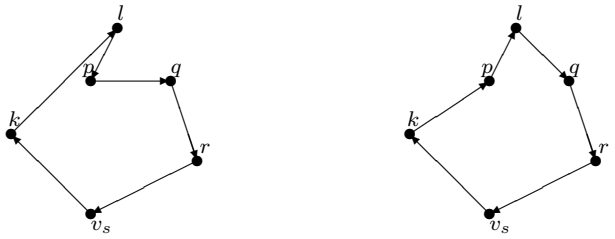
NNH algorithm for the TDTSP

Step 0:
Start at the depot and take $l = v_s$ at time t_s with $Q = \{1, 2, \ldots, n-1\}$ and $\mathcal{R} = \{v_s\}$.
Step 1:
while $Q \neq \emptyset$ **do**
Compute the arrival time at every possible neighbor of l in Q and find a vertex j with the shortest travel time.
Mark l as visited and take $l = j$, $Q := Q - j$ and $\mathcal{R} := \mathcal{R} \cup \{j\}$.
end while
Step 2:
If $Q = \emptyset$, take the route $\mathcal{R}$ which gives a path through all the vertices and add the travel time from the last vertex to the depot.

In Step 1 it is possible to compute all travel times because the starting time for the tour is known. We know the exact time we leave from the vertex l and the function $w_{ij}(t)$ gives the time we arrive at vertex j. The complexity of the NNH is $\mathcal{O}(n^2)$.

One disadvantage of this heuristic is that the travel time between the last customer and the depot could be very long. For the TSP the tour can be improved by an iterative process. Often 2-opt or k-opt are used, this means that in each step two or k arcs will be removed. The fragments will be reconnected by two or k new arcs, such that a new (and shorter) tour is created. If we apply this iterative process on the TDTSP it is possible that more arcs have to be changed because the arrival times will have changed. Consider an example in which we remove arc (k, l) and arc (p, q). Suppose that (k, l) is the first arc in the tour and that it would give a shorter tour if we submit arc (k, p). Then we have to calculate all the other arcs in the tour because the arrival times at vertices p, l, q, r have changed. See the figure below:

Figure 1



This shows that if an iterative process is applied after the NNH for the TDTSP, all the arcs have to be recalculated for the new tour. With these findings we see that it has no influence how many arcs are removed, so 2-opt could be a good process to use. Another way to improve the tour from the NNH is choosing in each step a different vertex after visiting the depot. We apply this method $n - 1$ times, for all the vertices except the depot, and choose the tour with the shortest travel time. In this case we know that the very long last arc will be chosen in one tour first and might be shorter.
This method gives the following heuristic algorithm for the TDTSP with order $\mathcal{O}(n^3)$

Iterative NNH algorithm for the TDTSP

Step 0:
Start at the depot $l = v_s$ at time t_s with $Q = \{1, 2, \ldots, n-1\}$, $k = 1$ and $\mathcal{R} = \{v_s\}$.
Step 1:
First choose the arc (v_s, k). t_k is the new starting time, mark v_s as visited and take $l = k$, $Q := Q - k$ and $\mathcal{R} := \mathcal{R} \cup \{k\}$
Step 2:
while $Q \neq \emptyset$ **do**
Compute the arrival time at every possible neighbor of l in Q and find a vertex j with the shortest travel time.
Mark l as visited and take $l = j$, $Q := Q - j$ and $\mathcal{R} := \mathcal{R} \cup \{j\}$.
end while

Step 3:
If $Q = \emptyset$, take the route $\mathcal{R}$ which gives a path through all the vertices and add the travel time from the last vertex to the depot to get tour $\mathcal{R}_k$ with time $T - t_s$.
Step 4:
Take $k = k + 1$ and repeat **Step 1-4** until $k = n - 1$, then go to **Step 5**.
Step 5:
Return the tour with minimal time $T - t_s$.

6.1.1 Quality of the heuristic

A heuristic gives hopefully a good feasible solution for a problem, but they give in general not the optimal solution. That is why one would like to know a priori how good the solution of a heuristic is. The *quality* of a heuristic H, given by $q(H)$, is defined by the worst-case solution of the heuristic. Let $H(I)$ be the worst-case solution of the heuristic, for instance I and let $O(I)$ be the optimal solution of the instance. In the case of a minimization problem we have that $H(I) \geq O(I)$. The quality of heuristic H is defined as $q(H) = \sup_P \frac{H(I)}{O(I)}$.
In most cases the optimal solution of an instance is not known, but there is a lower bound $L(I)$. The lower bound can be computed from the integer linear programing formulation of the problem. We give the formulation given by Bigras et al. [Bigr].
Let $G = (V, A)$ be a directed graph with the vertex set $V = 1, ..., n$. For each arc $(i, j) \in A$, the cost c_{ij}^t of traveling on the arc at period t is known, where $t = 1, 2, \ldots, T$. If the decision variable $x_{ij}^t = 1$ the arc (i, j) is traveled starting from vertex i in time period t. The TDTSP can be formulated as follows:

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{t=1}^{T} c_{ij}^t x_{ij}^t \tag{6.1}$$

$$\text{s.t.} \sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij}^t = 1, \quad t = 1, \ldots, T \tag{6.2}$$

$$\sum_{(1,j)\in A} x_{1j}^1 = 1 \tag{6.3}$$

$$\sum_{(i,1)\in A} x_{i1}^T = 1 \tag{6.4}$$

$$\sum_{(i,j)\in A} x_{ij}^t - \sum_{(i,j)\in A} x_{ij}^{t+1} = 0, \quad t = 2, \ldots, n-1, \ j = 1, \ldots, n \tag{6.5}$$

$$x_{ij}^t \in \{0, 1\} \quad , i = 1, \ldots, n, \ j = 1, \ldots, n, \ t = 1, \ldots, n \tag{6.6}$$

Constraint (6.2) gives that each vertex i is assigned to exactly one other vertex j. Constraint (6.3) and (6.4) imply that the depot is left from v_s at $t = 1$ and returned to v_s at $t = n$. Constraint (6.5) makes sure that if one enters a vertex one also will leave the vertex.
The solution of this linear programming formulation gives us the lower bound $L(I)$ for an given instance. So we have $L(I) \leq O(I) \leq H(I)$.
In the previous section we gave two algorithms, take $H_1(I)$ the worst-case solution of the first for NNH the TDTSP for an given instance I and $H_2(I)$ the worst-case solution of the iterative NNH for the TDTSP for the same instance. We know that the solution of the second heuristic is at least the same or better than the first one so $H_2(I) \leq (H_1(I)$ and $L(I) \leq O(I) \leq H_2(I) \leq H_1(I)$. We can calculate the gap between lower bound and the value of a feasible solution obtained by the heuristic. This gap measure is an upper bound of the difference between the value of the solution of the heuristic and an optimal solution.

7. Time dependent vehicle routing problem

Most of the models for the VRP that are described in the literature, like we noted in chapter 3.4, assume constant travel times. Clearly, ignoring the fact that the travel time between two customers does not depend only on the distance traveled, but also on the time of the day, has impact on the application of these models to real-world problems. The *time dependent vehicle routing problem* (TDVRP) has the same variants as the VRP but here the travel times depend on the time of day. The TDVRP has seldom been studied because they are harder to model and more difficult to solve than the VRP. We give a comprehensive overview about the literature we found on TDVRPs.

7.1 Literature

Malandraki and Daskin (1992) [Mala] were the first to formulate the TDVRP. They used a mixed integer linear programming formulation. In the formulation of the problem an $n \times n$ time dependent matrix $C(t) = c_{ij}(t_i)$ was used, which gives the travel times on an arc $(i, j) \in A$ as a function of the time t. In this matrix, $c_{ij}(t_i)$ is a step function of the time of the day with t_i at vertex i. Each arc (i, j) is replaced by M_{ij} parallel arcs, where $M_{ij})$ is the number of distinct time intervals considered in the step function $c_{ij}(t_i)$. The depot is expanded by K vertices $(n+1, \ldots, n+K)$ corresponding to returning depot vertices for each of the K vehicles. The aim is to minimize the total routing time (travel time plus service time plus waiting time). The following constraints are used:
K = number of vehicles
n = number of vertices including the depot
M = number of time intervals considered for each arc
c_{ij}^m = travel time from vertex i to j if starting at i during time interval m;
$c_{ii}^m = \infty \; \forall i, m$
c_i = service time at vertex im; $c_i = 0$ at the depot
T_{ij}^m = upper bound for time interval m for arc (i, j)

t = the starting time from the depots
b_k = weight capacity of vehicle k
d_i = weight to be collected at customer i
B_1, B_2 = a large number
$B = \max_k b_k$ = capacity of large vehicle
L_i = earliest time that the salesman can arrive at vertex i
U_i = latest time that the salesman can arrive at vertex i
t_j = departure time of any vehicle from vertex j
$w_j \geq$ weight that carried by a vehicle when departing from vertex j
x_{ij}^m is a decision variable and is 1 if any vehicle travels directly from vertex i to vertex j starting from i during time interval m ans is 0 otherwise.
The TDVRP may be formulated as follows:

$$\min \sum_{k=1}^{K} t_{n+k} \tag{7.1}$$

$$\text{s.t.} \sum_{i=1}^{n} \sum_{m=1}^{M} x_{ij}^m = 1, \quad j = 2, \ldots, n+K \tag{7.2}$$

$$\sum_{j=2}^{n+K} \sum_{m=1}^{M} x_{ij}^m = 1 \quad , i = 2, \ldots, n, \tag{7.3}$$

$$\sum_{j=2}^{n} \sum_{m=1}^{M} x_{ij}^m = K \tag{7.4}$$

$$t_1 = t \tag{7.5}$$

$$t_j - t_i - B_1 x_{ij}^m \geq c_{ij}^m + c_j - B_1 \tag{7.6}$$

$$t_i + B_2 x_{ij}^m \leq T_{ij}^m + B_2 \tag{7.7}$$

$$t_i - T_{ij}^{m-1} x_{ij}^m \geq 0 \tag{7.8}$$

$$for\ 7.6,\ 7.7,\ 7.8\ (i = 1, \ldots, n;\ j = 2, \ldots, n+K; i \neq j;\ m = 1, \ldots, M)$$

$$L_i + c_i \leq t_i \leq U_i + c_i \quad , i = 1, \ldots, n+K \tag{7.9}$$

$$w_j - w_i - B \sum_{m=1}^{M} x_{ij}^m \geq d_j - B \quad (i = 1, \ldots, n;\ j = 2, \ldots, n+K; i \neq j) \tag{7.10}$$

$$w_1 = 0 \tag{7.11}$$

$$w_{n+k} \leq b_k \quad (k = 1, \ldots, K) \tag{7.12}$$

$$x_{ij}^m \in \{0, 1\} \quad \forall i, j, m \tag{7.13}$$

$$t_i \geq 0,\ w_i \geq 0\ \forall i \tag{7.14}$$

In the formulation of the problem, waiting at a vertex is allowed. Constraints (7.2) to (7.4) ensure that each customer is visited exactly once and

that exactly K vehicles are used. Constraints (7.6) compute the departure time at vertex j. The objective function (7.1) ensures that this constraint is applied with equality when $x_{ij}^m = 1$ except in cases where waiting at j decreases the objective function value. The temporal constraints (7.7) and (7.8) ensure that the proper parallel arc m is chosen between vertices i and j according to the departure time from i. We want to model the following constraints: if $x_{ij}^m = 1$ then $T_{ij}^{m-1} \leq t_i \leq T_{ij}^m \ \forall i, j, m$. This means that t_i belongs to the time interval m defined by the above inequalities if the arc used corresponds to the same time interval m. We can set the large number B_2 equal to the latest possible return time of a vehicle. Constraints (7.9) impose the time windows that are defined in terms of the arrival times at the vertices. Constraints (7.10) to (7.12) impose the capacity restrictions. Constraint (7.11) states that all vehicles leave the depot empty. Constraint (7.10) ensure that the weight carried by the vehicle leaving customer j is at least equal to the weight when leaving the previously visited customer i plus the weight of the commodity picked up. Constraint (7.12) ensure that the capacity of each vehicle is not exceeded.

The problem does not take the FIFO-property into account. In the article, heuristics are given for the TDTSP and TDVRP without time windows. The algorithms are based on the nearest neighbor (greedy) heuristic (NNH) for the traveling salesman problem. For the TDVRP a heuristic sequential route construction is given. A new vehicle is only introduced when no customer can be assigned using the current vehicle(s). In this case the worst runtime performance is bounded by $\mathcal{O}(n^2M)$. Another heuristic, simultaneous route construction, is given. In this heuristic, a new vehicle is introduced, at any state of the algorithm, when this leads to the use of the shortest arc that does not violate the temporal requirements and the capacity of the vehicle. A solution can be obtained in $\mathcal{O}(n^2KM)$ time, with K being the number of vehicles. When the total capacity is much larger than the total demand, then one should avoid using this algorithm. In addition, a cutting plane heuristic algorithm for the TDTSP with a given starting time from the depot, and without time windows, is briefly discussed. The algorithms are tested on randomly generated problems with 10 to 25 vertices. The travel times are represented by step functions of two or three time periods per arc on average. The NNH requires very low computing times. The cutting plane algorithm is much more computationally expensive and solves only small problems but finds a solution better than, or at least as good as the solution obtained by the NNH in two of the three problems tested.

In 2003 Ichoua, Gendreau and Potvin [Icho] proposed a tabu search solution method in order to heuristically solve TDVRPs with soft time windows and which guarantees the FIFO-property. Like Malandraki and Daskin, this model uses a time-dependent matrix $C(t)$, with p different time periods. The speed of the vehicle changes when the boundary between two consecutive time periods is crossed. Because the travel speed is a step function, the

travel time is a piece-wise continuous function over time. The objective is to minimize a weighted sum of total distance traveled and total lateness over all customers. The algorithm uses parallel tabu search developed by Taillard (1997). The method is tested using Solomon's 100 customer problems. In these problems customer locations are generated randomly and uniformly within a $[0, 100]^2$ square. The experiments were performed to evaluate the model in a static and dynamic environment. Three time periods and three types of time dependent arcs are studied. The results of their experiments show that the time-dependent model provides very significant improvements to the objective value over the model with fixed travel times, thus indicating the usefulness of additional information about the problem.

Fleischmann, Gietz and Gnutzmann (2004) [Flei] use traffic information of Berlin to see whether they could improve vehicle routing and scheduling. They present a general framework for the implementation of time-varying travel times in various vehicle routing algorithms that were already proposed in the literature. The goal was to minimize the number of vehicles and total travel time. Computational tests based on the traffic in Berlin shows that the use of constant average travel times can lead to significant underestimation of the actual total travel times.

More recently, Van Woensel, Kerbache, Peremans, and Vandaele (2008) [Woen] used a tabu search algorithm to heuristically solve the TDVRP. They used approximations based on queuing theory to determine the travel speeds. The objective is to find the minimum cost vehicle routes. Three different speed approaches are compared: no speed effects, three time periods (morning, mid-day and evening) and a queuing model with 144 time intervals. A dataset with real-life observations is used as input. For each of the three speed approaches there are solutions generated using the tabu search heuristic. These solutions are re-evaluated using a different validation dataset for a different comparable day. The results show that the total travel time can be improved significantly, when explicitly taking congestion into account during the optimization. Additionally the authors found that a higher number of time zones improves the solution quality. Another finding is that adapting a starting time for a solution has significant effects on the obtained solution quality. The extra computing time for large data sets is significant but is worthwhile because the solution quality is improved.

Soler, Albiach and Martínez (2008) [Sole] use a different approach because they transform the TDVRP with time windows through several steps into an asymmetric capacitated VRP, a well-known routing problem. This problem can be solved optimally and heuristically with known codes. The aim is to minimize the sum of the costs of the different routes given that the traversing arc times satisfy the FIFO-property. They provide a way to optimally solve the problem, at least for small size instances (due to its complexity).

An alternative way is described by Donati et al. (2008) [Dona]. The algorithm that is used, is based on the ant colony system and local search

procedures. The algorithm is named MACS-VRPTW. They describe a TD-VRP with delivery time windows. The basic idea of ant colony optimization is to use a positive feedback mechanism to reinforce those parts that belong to a good solution, while discarding those that belong to poor solutions. This is done with the possibility to temporally store this information so that it will be locally available to all the individuals. They combine two algorithms, one to optimize the number of vehicles and one for optimizing the total travel time. The algorithms are supported by local search procedures that store and update the slack times or feasible delays. The algorithms are tested using some variations of the Solomon problems and on a real-life network in Padua, Italy.
Last year Figliozzi (2009) [Figl] presented a new solution approach, an iterative route construction and improvement algorithm (IRCI), for the TD-VRP with hard and soft time windows. The improvements are obtained at a route level and do not rely on any type of local improvement procedure. The solution algorithms can handle constant speed at time-dependent speed problems. A new formulation for the TDVRP with soft and hard time windows is presented. The travel speed in any arc is a positive and continuous function of time, which guarantees the FIFO-property. The travel times may be asymmetric and waiting time at a customer is allowed. The primary objective function is the minimization of the number of routes; the optimal number of routes is unknown. A secondary objective is the minimization of the total time or distance. The solution method to minimize fleet size is divided into two phases and heuristics: route construction and route improvement. For the construction heuristic a route building heuristic is presented which will be repeatedly executed. The route building heuristic is a generalized nearest neighbor heuristic. Similarly the route construction heuristic is repeatedly executed during the improvement heuristic. For any given route a dynamic programming approach can be used to determine the optimal service start times for customers. The performance of the IRCI was tested on the Solomon instances with constant travel speed and on a time dependent variant. The IRCI is faster than the methods presented by Donati et al. (2008) but in terms of solution quality the solutions are about 4% less than the best results ever obtained by the Solomon instances. For the time dependent cases the author did not find similar results, so a comparison was not possible.

7.2 A heuristic for the TDVRP

In 2008 Soler et al. described an idea to solve the TDVRP optimally, by transforming it into an asymmetric capacitated VRP. The consequences of their approach is that the asymmetric capacitated VRP is still a hard problem to solve. These transformations will make the problem size blow up. It

would only be possible to do this for small instances. The article is overall theoretical, and to the best of our knowledge, no computational experiments using this approach have been reported so far. To find an optimal solution for the TDVRP is not yet a realistic option for real-world instances.
We give an approach for a heuristic based on the heuristic for the TDTSP we presented in Chapter 6.1. The TDVRP can be seen as the problem of solving the TDTSP for each vehicle, if the customers are divided through out the vehicles. In Chapter 3.4.1 we have seen that the customers can be divided with a sweep method, which seems to be a good idea. The problem in the time dependent case is that some vehicles could have routes involving heavy congestion. It is thus very important to divide the customers in a "good" way over the vehicles. We try to divide the customers in three groups, the customers with less rush hour traffic in the morning or mid-day or evening. If there is a congestion in one way from customer i to j, we advice to place customer i in a later time period than customer j, so the congestion can be avoided. When the selection is made the algorithm in chapter 6.1 can be applied to each separate vehicle.

8. Conclusion

In this master thesis we have studied optimization problems in networks with time dependent arcs. The aim is to describe real world problems in a more realistic way than the optimization problems with constant arcs.
First we looked at the time dependent shortest path problem. We gave an algorithm for this problem where the starting time is known and holds the FIFO-property. The algorithm is based on the Dijkstra algorithm for the shortest path problem. We gave a proof of the algorithm because it could not be found in the literature. If the starting point of the time dependent shortest path has to lie in a given starting time interval, there are two approaches given to handle this question. For both methods the complexity of the algorithm could be unbounded because it depends on the arrival time functions or the number of time intervals which can be exponential.
Secondly, time dependent flows are examined. The first algorithm ever to find a maximal flow over time, by Ford and Fulkerson, is given. To also take congestion into account, a time-expanded network that contains time layers is explained. On this network an algorithm for a static flow can be applied. The time-expanded network grows with the number of time intervals so it could get very large.
Another problem we studied is the time dependent extension of the traveling salesman problem. We gave a nearest neighbor heuristic to achieve a tour for a given starting time. The NNH could give a much different tour from the optimal solution, that is why we improve the tour by iteratively applying the NNH. There will be $n - 1$ tours calculated and the shortest tour in time is chosen.
Finally, the time dependent vehicle routing problem is discussed by giving a comprehensive literature overview and suggestions for solving the problem. There is almost no information known to derive an optimal solution. Last year Figliozzi described a heuristic for this problem and gave some results. To evaluate this heuristic theoretically and computationally seems an interesting avenue for future research. We like to recommend this for future research. Furthermore, it would be interesting to look at the time windows of the TDVRP and to study the problems from this thesis for a given starting time interval and see if it is possible to formulate polynomial heuristics that could produce good-quality solutions for realistic instances.

Appendix A

FIFO: First In First Out
NNH: Nearest Neighbor Heuristic TDSP: Time Dependent Shortest Path
TDTSP: Time Dependent Traveling Salesman Problem
TDVRP: Time Dependent Vehicle Routing Problem
TSP: Traveling Salesman Problem
VRP: Vehicle Routing Problem
VRPTW: Vehicle Routing Problem with Time Windows
IRCI: Iterative Route Construction and Improvement algorithm

Bibliography

[Ahuj] Ahuja, R.K., Magnanti, T.L., Orlin, J.B. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall (1993)

[Appl] Applegate, D.L., Bixby, R.E., Chvátal, V., Cook, W., Espinoza, D.G., Goycoolea, M., Helsgaun, K. Certification of an optimal TSP tour through 85,900 cities. *Operations Research Letters, volume 37(1)*, 11-15 (Jan 2009)

[Appl2] Applegate, D.L., Bixby, R.E., Chvátal. V., Cook, W.J. *Traveling Salesman Problem: A Computational Study.* Princeton University Press (2006)

[Aror] Arora, S. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *Journal of the ACM, volume 45(5)*, 753-782 (Sep 1998)

[Asad] Asadpour, A., Goemans, M.X., Madry, A., Oveis Gharan, S., Saberi, A. An $\mathcal{O}(\log n/\log\log n)$-approximation algorithm for the asymmetric travelings salesman problem. *Proceedings of* 21^{ste} *ACM-SIAM Symposium on Discrete Algorithms* (2010)

[Ball] Ball, M.O., Magnanti, T.L., Monma, C.L., Nemhauser, G.L. *Network Routing.* Elsevier science B.V. (1995)

[Bert] Bertsimas, D., Tsitsiklis, J.N. *Introduction to Linear Optimization.* Athena Scientific, Belmont, Massachusetts (1997)

[Bell] Bellman, R. On a routing problem. *Quarterly of Applied Mathematics, volume 16(1)*, 87-90 (1958)

[Bigr] Bigras, L.P., Gamache, M., Savard, G. The time-dependent traveling salesman problem and single machine scheduling problems with sequence dependent setup times. *Discrete Optimization, volume 5(4)*, 685–699 (Nov 2008)

[Chri] Christofides, N. Worst-case analysis of a new heuristic for the traveling salesman problem. *Technical Report 388. Graduate School of Industrial Administration, Carnegic Mellon University, Pittsburgh* (1976)

[Dant] Dantzig, G.B., Ramser, J.H. The truck dispatching problem. *Management Science, volume 6(1)*, 80-91 (1959)

[Dan2] Dantzig, G.B., Fulkerson, R., Johnson, S.M. Solution of a large-scale traveling salesman problem. *Operations Research 2(4)*, 393-410 (Nov 1954)

[Dehn] Dehne, F., Omran, M.T., Sack, J-R. Shortest paths in time-dependent FIFO networks using edge load forecasts. *Proceedings of the Second International Workshop on Computational Transportation Science* (nov 2009)

[Dijk] Dijkstra, E.W. A note on two problems in connection with graphs. *Numerische Mathematik, volume 1*, 269-271 (Oct 1959)

[Ding] Ding, B., XU Yu, J., Qin, L. Finding time-dependent shortest paths over large graphs. *Proceedings of the 11th international conference on Extending database technology*, 205-216 (2008)

[Dona] Donati, A.V., Montemanni, R., Casagrande, N., Rizzoli, A.E., Gambardella, L.M. Time dependent vehicle routing problem with a multi ant colony system. *European Journal of Operational Research, volume 185(3)*, 1174-1191 (2008)

[Drey] Dreyfus, S.E. An appraisal of some shortest path algorithms. *Operations Research, volume 17(3)*, 395-412 (1969)

[Even] Even, S. *Graph Algorithms.* Computer science press (1979)

[Figl] Figliozzi, M.A. A route improvement algorithm for the vehicle routing problem with time dependent travel times. *Proceedings of the 88th Transportation Research Board Annual Meeting, Washington DC. USA* (Jan 2009)

[Flei] Fleischmann, B., Gietz, M., Gnutzmann, S. Time-varying travel times in vehicle routing. *Transportation Science, volume 38(2)*, 160–173 (2004)

[Ford] Ford Jr, L.R. Network flow theory. *The Rand Corporation Paper, Santa Monica*, 923 (Aug 1956)

[Ford1] Ford Jr, L.R., Fulkerson, D.R. Maximal flow through a network. *Canadian Journal of Mathematics, volume 8*, 399-404 (1956)

[Ford2] Ford Jr, L.R., Fulkerson, D.R. Constructing maximal dynamic flows from static flows. *Operations Research, volume 6*, 419-433 (1958)

[Ford3] Ford Jr, L.R., Fulkerson, D.R. *Flows in Networks.* Prinston University Press, Prinston, N.J. (1962)

[Fuka] Fukasawa, R., Longo, H., Lysgaard, J., Poggi de Aragão, M., Reis, M., Uchoa, E., Werneck, R.F. Robust branch-and-cut-and-price for the capacitated vehicle routing problem. *Mathematical Programming, volume 106(3)*, 491-511 (2006)

[Gare] Garey, M.R., Johnson, D.S. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* WH Freeman & Co, New York (1979)

[Icho] Ichoua, S., Gendreau, M., Potvin, J-Y. Vehicle dispatching with time-dependent travel times. *European Journal of Operational Research, volume 144(2)*, 379-396 (Jan 2003)

[Kohl] Köhler, E., Möhring, R.H., Skutella, M. Traffic networks and flows over time. *Proceedings of the 13th annual ACM-SIAM symposium on discrete algorithms, Society of Industrial and Applied Mathematics*, 174-183 (2002)

[Koh2] Köhler, E., Skutella, M. Flows over time with load-dependent transit times. *Proceedings of the 13th annual ACM-SIAM symposium on discreet algorithms, San Francisco, CA*, 174-183 (2002)

[Mala] Malandraki, C., Daskin, M.S. Time dependent vehicle routing problems: formulations, properties and heuristic algorithms. *Transportation Science, volume 26(3)*, 185-200 (Aug 1992)

[Mal2] Malandraki, C., Dial, R.B. A restricted dynamic programming heuristic algorithm for the time dependent traveling salesman problem. *European Journal of Operational Research, volume 90(1)*, 45-55 (1996)

[Mira] Miranda-Bront, J.J., Mendez-Diaz, I., Zabala, P. An integer programming approach for the time dependent traveling saleman problem. *Electronic Notes in Discrete Mathematics, volume 36*, 351-358 (Aug 2010)

[Orda] Orda, A., Rom, R. Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length. *Journal of the ACM, volume 37(3)*, 607-625 (1990)

[Pica] Picard, J.-C., Queyranne, M. The time-dependent traveling salesman problem and its application to the tardiness problem in one-machine scheduling. *Operations Research, volume 26(1)*, 86-110 (1978)

[Skut] Skutella, M. An introduction to network flows over time. *Cook, W., Lovász, l., and Vygen, J.: Research Trends in Combinatorial Optimization. Springer, Berlin*, 451-482 (2009)

[Sole] Soler, D., Albiach, J., Martinez, E. A way to optimally solve a time-dependent vehicle routing problem with time windows. *Operations Research Letters, volume 37(1)*, 37-42 (2009)

[Sung] Sung, K., Bell, M.G.H., Seong, M., Park, S. Shortest paths in a network with time-dependent flow speeds. *European Journal of Operational Research, volume 121(1)*, 32-39 (Feb 2000)

[Vand] Vander Wiel, R.J., Sahinidis, N.V. An exact solution approach for the time-dependent traveling salesman problem. *Naval Research Logistics, volume 43(6)*, 797–820 (Sep 1996)

[Woen] Woensel van, T., Kerbache, L., Peremans, H., Vandaele, N. Vehicle routing with dynamic travel times: A queueing approach. *European Journal of operational Research, volume 186(3)*, 990-1007 (May 2008)