Abstract

Title of Thesis: THE QUICK TIME DEPENDENT QUICKEST FLOW PROBLEM: A LESSON IN ZERO-SUM CYCLES

Degree Candidate: Harsh Dhundia

Degree and year: Masters of Science, 2004

Thesis directed by: Dr. Elise Miller-Hooks
Department of Civil and Environmental Engineering

A quick solution technique for the integral time-dependent quickest flow problem with no waiting is presented. The proposed technique is based on the successive shortest path approach and modifies an existing algorithm to improve its average performance. At each iteration, a reoptimization procedure is employed to determine the augmenting path given updates to the residual graph. The residual graph, by construction, almost always contains zero-sum cycles when employed in this context. These zero-sum cycles pose a unique problem for the reoptimization technique. A heuristic that can be embedded in the reoptimization algorithm to provide path solutions in the presence of zero-sum cycles has been proposed. In the computational experiments, the heuristic provided an optimal solution nearly 100% of the times. Further, a modified implementation of an existing path-finding algorithm has been used to solve the time-dependent quickest flow problem with source waiting.

The Quick Time Dependent Quickest Flow Problem: A Lesson in Zero-Sum Cycles

by

Harsh Dhundia

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Master of Science
2004

Acknowledgements

I would like to express my gratitude to Dr. Elise Miller-Hooks for her continued support and patience. This work would not have been possible without her encouragement and guidance. She instilled in me the spirit to conduct research.

Ruchi, Roger, Dan, Petru, Samer, Rahul, Sathaporn and Stacy deserve a special note of thanks for all their love and support.

Finally, I would like to thank my parents for being all they have been to me. I could not have done anything without their support, love and affection.

Table of Contents

Chapter 1: Introduction

In this thesis, a heuristic solution technique for the integral time-dependent quickest flow problem (*TDQFP*) is presented. The *TDQFP* seeks to send a given amount of supply from a source to a sink such that the last unit of supply arrives at the sink in the minimum possible time. Attributes of the network (availability of supply at the source, arc travel times and arc capacities) vary with time, i.e. the network is time-dependent. Further, the capacities on the arcs are recaptured over time, i.e. a dynamic network flow problem is considered.

The proposed heuristic builds on the pseudopolynomial time TDQFP algorithm of Miller-Hooks and Stock Patterson (2004) that was designed to solve the *TDQFP* when unlimited waiting is allowed at all nodes. The TDQFP algorithm is based on the successive shortest path approach (See Ahuja et al. 1993 for background on successive shortest path approach for the static problem) and is, thus, an iterative algorithm. Like all algorithms based on the successive shortest path approach, it relies on a path-finding mechanism at each iteration. In the computational results reported by Miller-Hooks and Stock Patterson (2004), it may be seen that their path-finding algorithm consumes approximately 67% of the total run time of the TDQFP algorithm on average.

In this thesis, the path-finding technique of the TDQFP algorithm was replaced with a path-reoptimization technique to provide quicker results for the *TDQFP* that also prohibited waiting. The reoptimization procedure used for this purpose was required to produce optimal results in the presence of zero-sum cycles (a cycle for which the weights on the constituent arcs sum to zero) in the graph. An existing exact reoptimization technique (Miller-Hooks and Yang, 2004) was modified to create a heuristic to meet this requirement of quickly producing optimal results in presence of zero-sum cycles. The primary contributions of this work are the introduction of the rationale and the specific steps for speeding up the TDQFP algorithm of Miller-Hooks and Stock Patterson (2004) and a heuristic solution to the zero-sum cycle problem encountered in the context of reoptimizing shortest paths specifically designed for application to the successive shortest path approach. The technique developed is referred to as the Quick-TDQFP heuristic.

The literature is replete with applications that have been modeled as quickest flow problems (QFPs). Production-distribution, fleet management, scheduling with deferral costs are a few examples of applications that have been posed and solved as QFPs in a time-invariant environment (Ahuja et al. 1993; Aronson, 1989). While problem attributes are often time-varying, relatively few works have addressed QFPs in a time-varying environment. Klingman and Mote (1982) formulated a multiperiod production-distribution problem as a minimum cost network flow problem (MCNFP) in a time-varying environment. The MCNFP is a variant of the QFP where travel cost instead of travel time is considered in the objective function. However, in their work the arc attributes remained fixed over time and only the node requirements were allowed to change. Opasanon (2004) solved a variant of the *TDQFP* to address an evacuation problem where the arc travel times were time-varying and the arc capacities were stochastic and time-varying.

In this thesis, a QFP is considered in the context of evacuating a large building or a geographic region. Because conditions inherent in circumstances warranting an evacuation change over time, it is critical when modeling this application that the time-varying nature of the network, in terms of arc traversal times and capacities be explicitly considered. It must be noted that the evacuation problem has multiple sources and multiple sinks while the *TDQFP* has a single source and a single sink. An efficient technique to convert dynamic network flow problems with multiple sources and sinks into single source, single sink problems can be found in Miller-Hooks and Stock Patterson (2004). Given such a technique exists, without loss of generality, for the remainder of this thesis only a single source, single sink dynamic network flow problem is considered.

A dynamic network flow problem involves movement of flow through the network over time. Each arc has a travel time and capacity associated with it. Arc travel times determine how long it takes for each unit of flow to get from the beginning node of the arc to the terminal node. Arc capacities restrict the amount of flow that can enter an arc per unit time[1]. Ford and Fulkerson (1962) provided a polynomial time solution to the maximum dynamic network flow problem, which seeks to send the maximum units of flow from a single source to a single sink within a given time bound. Burkard et al. (1993) provided efficient polynomial algorithms for the quickest flow problem. A polynomial algorithm for the integral quickest transshipment problem (extension of the QFP for multiple sources and multiple sinks) was proposed by Hoppe and Tardos (2000).

A relatively small amount of research has been conducted on solving dynamic network flow problems in a time-dependent environment. Though Halpern (1979) and Anderson et al. (1982) considered time-varying capacities, the travel times remained fixed in their studies. Cai et al. (2001) considered time-varying network attributes (i.e. both arc travel times and arc capacities are time-varying) in a pseudopolynomial time

[1] This approach to modeling release of capacity over time is in line with that in Miller-Hooks and Stock Patterson (2004).

algorithm that was developed to solve the time-dependent minimum cost flow problem. They addressed the problem with (a) no-waiting, (b) limited waiting and (c) unlimited waiting allowed at all nodes in a non-FIFO (non-first-in-first-out) network with discrete time intervals. Miller-Hooks and Stock Patterson (2004) introduced the time-dependent version of the integral quickest flow problem and proposed the TDQFP algorithm, a pseudopolynomial time algorithm to solve the case when unlimited waiting is permitted at all nodes in a non-FIFO network with discrete time intervals. Note that the algorithms presented in both Cai et al. (2001) and Miller-Hooks and Stock Patterson (2004) have pseudopolynomial time complexity. Thus, in time-dependent networks, an approach that can provide quicker solutions than can these techniques is of interest.

The heuristic technique proposed in this thesis builds on the TDQFP algorithm of Miller-Hooks and Stock Patterson (2004). At each iteration of the TDQFP algorithm, flow is assigned to a path (an augmenting path) that permits arrival at the sink at the earliest time. This path is determined by using the TDEAT (Time-Dependent Earliest Arrival Time) algorithm proposed by Miller-Hooks and Stock Patterson (2004) for this purpose. The network is then updated to incorporate information about this flow. Such a network is called a time-dependent residual network[2]. Updating the residual network involves the calculation of residual capacities on the arcs that were a part of the augmenting path and addition of backwards arcs when required to permit the return of capacity to an arc and reverse a decision from the previous iteration. Once the residual network has been updated, a new iteration commences. The process is repeated until either no more augmenting paths can be found or all the supply has been shipped. Cai et

[2] See Miller-Hooks and Stock Patterson (2004) for additional detail on updating time-dependent residual networks

al. (2001) employ a similar procedure to address the minimum cost time-dependent network flow problem.

In the computational results reported by Miller-Hooks and Stock Patterson (2004), it may be seen that the TDEAT algorithm consumes between 50% and 90% of the total run time of the TDQFP algorithm. Similarly, the path-finding algorithm used for the SEscape algorithm (Opasanon, 2004), which has a structure similar to that of the TDQFP algorithm, consumes 99% of the computational time on average. Thus, in a time-variant environment, for algorithms based on the successive shortest path approach, a technique that would reduce the time spent in seeking augmenting paths could significantly improve the overall run time of the algorithm.

The path-finding algorithm is called at the beginning of each iteration of the TDQFP algorithm to determine a new augmenting path in a graph in which attributes for only a few arcs have changed. This means that the problem of path-finding at each iteration of the algorithm may be seen as a problem of updating the path given a few changes to the network. Thus, instead of re-solving for the paths beginning from a scratch, using a reoptimization procedure that could quickly return an updated augmenting path at each iteration, could result in savings in computation time. The Quick-TDQFP heuristic is based on this idea.

In the context of evacuation from a building or a region, solutions that require waiting at an intermediate location are not likely to be adopted. Thus, the TDQFP algorithm in its present form cannot be applied to solve the evacuation problem. A technique that will generate paths that do not mandate waiting is required i.e., the solution technique should solve the *TDQFP* with no-waiting. Two variations of the no-waiting problem have been addressed in this thesis: one where waiting is not allowed at any of the nodes and the other where waiting is allowed only at the source node. The TDQFP algorithm can be adapted to solve these problems with no waiting by incorporating an appropriate time-dependent least-time path problem in place of the TDEAT (Time-Dependent Earliest Arrival Time) algorithm proposed in Miller-Hooks and Stock Patterson (2004) that was specifically developed to take advantage of the fact that waiting was permitted at all nodes.

Of the many available algorithms to solve the time-dependent least-time path problem (Cooke and Halsey, 1966; Dreyfus, 1969; Orda and Rom, 1990; Ziliaskopoulos and Mahmassani, 1993; Chabini, 1998; Pallottino and Scutella, 1998, Cai et al. 2001, Miller-Hooks and Stock Patterson, 2004), only a few might be useful in the context of this thesis. An algorithm that can be used to produce optimal paths in the presence of arcs with negative travel times in a time-dependent non-FIFO network where time is assumed to be discretized into small intervals and no waiting is allowed at any node is required. Because the label-setting techniques (Dreyfus, 1969; Chabini, 1998; Pallottino and Scutella, 1998) require non-negative arc weights, only label correcting algorithms (see Ahuja et al. 1993 for details on label-correcting and label-setting algorithms) will be applicable. Orda and Rom (1990) provided algorithms that could not provide solutions when waiting was not allowed at any node making them unsuitable for use in this thesis.

Specialized algorithms to determine augmenting paths in time-dependent residual graphs were proposed by Miller-Hooks and Stock Patterson (2004) and Cai et al. (2001). The TDEAT algorithm of Miller-Hooks and Stock Patterson (2004) solves for the Time-dependent Earliest Arrival Time Path in a residual graph from a single source to all nodes for a given departure time from the source. It was designed specifically to take advantage of the fact that waiting is allowed at all nodes and therefore it cannot be used in the context of this thesis.

Cai et al (2001) proposed the SDFP-ZW algorithm, which finds the augmenting paths when no waiting is allowed at any of the nodes in a time-dependent non-FIFO network. The SDFP-ZW algorithm determines the shortest (least-cost) paths from a specified source node to all other nodes in the graph for a specified departure time from the source. The algorithm is based on two different search operations, one each for arcs with negative and positive travel times. The algorithm has an initialization step of $O(n \cdot I)$, a sorting step that can best be implemented in $O(m \cdot I)$, and then the main iterative body of the algorithm which has a complexity of $O(n^3 \cdot I^2)$, where m is the number of arcs in the graph, n is the number of nodes, and I is the number of discrete time intervals in the period of interest.. Though the algorithm is stated to have a worst-case computational complexity of $O(m \cdot n \cdot I^2)$, when using it for determining augmenting paths in the successive shortest path framework, the initialization and sorting steps will have to be implemented at each iteration. This makes the algorithm very inefficient for use in solving the *TDQFP* using the successive shortest path framework. Not only is the technique inefficient, it is hard to understand and implement. This makes the algorithm undesirable in this work where we seek to improve the computational time of a technique to solve the *TDQFP*.

Of interest in this thesis is the TDLTP algorithm proposed by Ziliaskopoulos and Mahmassani (1993), which is closely related to the work by Cooke and Halsey (1966). The TDLTP algorithm is a label-correcting algorithm that solves for the Time-dependent

Least Time Paths (TDLTP) from all nodes to a given destination for all departure times in a discretized non-stationary time period of interest. After the period of interest, the arc travel times are assumed to be stationary with values equal to those in the last time interval. Waiting is not permitted at any of the nodes and the network is allowed to be non-FIFO. Although the algorithm was initially proposed only for non-negative arc travel times, it can produce optimal results with negative arc travel times as well. It has a worst-case computational complexity of $O(n^3 I^2)$ where n is the number of nodes in the network and I is the number of discrete time intervals considered.

Furthermore, Miller-Hooks and Yang (2004) showed how the TDLTP algorithm can be modified to create its reoptimization version for use when multiple heterogeneous arc travel time updates are received. However, the TDLTP Reoptimization algorithm of Miller-Hooks and Yang (2004) is not readily applicable in this work, because it was proposed to provide solutions only for positive real-valued arc travel times. When working in residual graphs, backwards arcs with negative travel times are introduced at each iteration. Coupled with existing arcs with positive travel times, the backwards arcs often form zero-sum cycles in the graph. While the TDLTP Reoptimization algorithm can provide optimal solutions in the presence of arcs with negative travel times, it fails if the graph has zero-sum cycles.

It is known that Bellman's equations are not sufficient to compute optimal shortest path labels in the presence of zero-sum cycles (Ahuja et al. 1993 and Bertsekas, 1991). However, many shortest path algorithms (eg. Dijkstra, 1958[3]; Bellman,1958; Ziliaskopoulos and Mahmassani, 1993; TDEAT algorithm of Miller-Hooks and Stock

[3] Note that Dijkstra's algorithm requires non-negative arc weights. In this case a zero-sum cycle would be composed of arcs, all of which have zero arc weight.

Patterson, 2004) that use Bellman's equations as their sole condition for optimality can provide optimal results for graphs with zero-sum cycles. It was observed in this research that such algorithms can produce optimal solutions because the optimality conditions are implemented in such a way that at each step the solution is ensured to be free of a zero-sum cycle. Specifically, this is accomplished by updating the labels if and only if the label can be improved. The TDLTP Reoptimization algorithm fails in the presence of zero-sum cycles because no check is made to ensure that the solution is a tree and does not contain a zero-sum cycle[4]. In this thesis, the TDLTP Reoptimization algorithm has been modified to create a heuristic by the introduction of an additional function employed just prior to updating a label so that the solution does not contain a zero-sum cycle. Though the additional function guarantees that there will be no zero-sum cycles in the solution, it terminates with sub-optimal solutions in some cases. The function is referred to as the path diving function and is discussed in Chapter 3.

Since, the *TDQFP* is a single source, single sink problem, only a path from one source to one sink is sought during each iteration. Thus, the TDLTP algorithm does a lot of extra work because it solves the all-to-one (See Ahuja et al. 1993 for details on all-to-one and one-to-all algorithms) problem for every departure time in the period of interest. Though a reoptimization technique has been proposed to improve the computational performance of the TDQFP algorithm when used with the TDLTP algorithm, its use does not guarantee an optimal solution. An exact algorithm that could provide qiuick results is desirable. In this light, a modified implementation of the Chrono-SPT paradigm developed by Pallotino and Scuttella (1998) is proposed here. The Chrono-SPT algorithm

[4] It must be emphasized that the TDLTP Reoptimization algorithm was proposed only for positive travel times and in that case no zero-sum cycles are possible.

solves the one-to-all problem for a single departure time from the source. The implementation technique proposed in this thesis is called the Multiple Chrono-SPT (M-Chrono-SPT) algorithm and has the ability to deal with negative arc travel times which the original Chrono-SPT did not. The Chrono-SPT algorithm allows source-waiting, that is, the algorithm determines the optimal departure time to leave the source to arrive at the sink at the earliest possible time by incurring the least possible travel time. The TDQFP algorithm when implemented with the M-Chrono-SPT algorithm as the path-finding algorithm solves the *TDQFP* with source-only waiting.

While the computational complexity of the Chrono-SPT algorithm is $O(mI)$, the M-Chrono-SPT has a worst case complexity of $O(m^2I^2)$ where m is the number of arcs in the network and I is the number of discrete time intervals. The extra computational effort for M-Chrono-SPT results from the steps employed to handle negative arc travel times.

Computational experiments were run on test networks used in Miller-Hooks and Stock Patterson (2004). The average in-degree and out-degree for each node in all the networks was approximately 4, with a minimum of 2 and a maximum of 9. The number of time intervals varied between 60 and 240. The algorithms were tested and compared for two states of supply load in the network i.e., light and heavy. The details of these experiments can be found in Chapter 4.

In the computational experiments, the solutions with the path diving heuristic were always found to be optimal. This can be attributed to the topology of the test network. The case where the heuristic fails to determine an optimal solution has been discussed in Chapter 3.The experiments revealed that, depending on the network size, on average, the run times of the Quick-TDQFP heuristic were four to eight times better than the TDQFP algorithm with TDLTP as the path-finding function. The improvement in run times was observed to be strongly correlated to the number of iterations required to finish shipping the supply. On implementing the TDQFP algorithm with the M-Chrono-SPT algorithm as the path-finding algorithm, it was found that the algorithm performs much better than its average computational complexity. For the set of networks used in the experiments, it was observed that the number of iterations (paths) required to ship all supply in the source-only waiting case was always less than of that in the no-waiting case.

Also, the number of iterations required by Quick-TDQFP heuristic to finish shipping all supply was slightly higher than the number of iterations required by the TDQFP algorithm. This can be attributed to the fact that in case of Quick-TDQFP heuristic where paths are updated at each iteration, there is a high probability that a section of the updated path will be the same as the old one. Since, a certain portion of the capacity of this path had already been used, it might not be possible to assign all the remaining supply for the particular departure time to this path. Thus, more paths will have to be sought.

The contributions of this work are the (a) development of the rationale of using reoptimization procedures to speed up the TDQFP algorithm, (b) identification of the zero-sum cycle problem for reoptimization in this context and (c) the provision of a heuristic solution for reoptimizing shortest paths in presence of zero-sum cycles by modifying the TDLTP Reoptimization algorithm. Moreover, an existing path-finding algorithm has been modified to provide optimal shortest path solutions given a network with negative arc weights.

In the next chapter, Chapter 2 the specific details of the Quick-TDQFP heuristic are provided and the important differences between the TDQFP algorithm and the Quick-TDQFP heuristic are described. In Chapter 3, the path-finding algorithms and path reoptimization in the presence of zero-sum cycles is discussed. Experimental results in terms of the run times of the algorithms are presented and analyzed in Chapter 4. The conclusions are reported in Chapter 5.

Chapter 2: The Quick-TDQFP Heuristic

2.1 Introduction

This chapter describes the Quick-TDQFP heuristic by characterizing how it differs from the original TDQFP algorithm of Miller-Hooks and Stock Patterson (2004). The steps of original TDQFP algorithm are provided in Appendix A. In section 2.2, an overview of the Quick-TDQFP heuristic is given and in section 2.3 the specific steps of the algorithm that differ from those in the TDQFP algorithm are presented and discussed.

2.2 The Quick-TDQFP heuristic

The Quick-TDQFP heuristic is based on the TDQFP algorithm of Miller-Hooks and Stock Patterson (2004), which was designed to solve the Time-Dependent Quickest Flow Problem (*TDQFP*) for the case when waiting is allowed at all nodes. A definition and mathematical formulation for the *TDQFP* was provided by Miller-Hooks and Stock Patterson (2004). They permit waiting at nodes by inclusion of self-loops with unit travel time at each departure time. Exclusion of these self loops from the set of arcs in the graph would result in the problem where no waiting is allowed at any of the nodes, since the flow conservation constraints would force the supply to move from the nodes at each time instance. Miller-Hooks and Stock Patterson (2004) show that the TDQFP algorithm produces optimal results with respect to their mathematical formulation, and it may be easily verified that the same would be true if the TDQFP algorithm were to solve a problem with no-waiting. The Quick-TDQFP heuristic focuses on the *TDQFP* when waiting is not allowed at any of the nodes. A similar methodology could be applied to the *TDQFP* with waiting.

Like the TDQFP algorithm on which it is based, the Quick-TDQFP heuristic uses the successive shortest path approach. Thus, at each iteration of the algorithm, flow is assigned to a path (called the augmenting path) such that the flow reaches the sink at the earliest time (i.e., an earliest-arrival-time-path is used). The residual graph is updated accordingly. Updating the residual network involves calculation of residual capacities on the arcs that were a part of the augmenting path and addition of backwards arcs where required to permit the return of capacity to an arc and reverse a decision from the previous iteration.

The difference between Quick-TDQFP heuristic and the TDQFP algorithm lies in the techniques used to determine the augmenting paths. The TDQFP algorithm finds the augmenting path at each iteration by determining a new augmenting path from scratch. On the contrary, the Quick-TDQFP heuristic employs a reoptimization technique to update the path from the prior iteration given changes to the network attributes. That is, it employs a path-finding function that heuristically updates the augmenting path from the last iteration through reoptimization. This function is called REOPT herein.

Updates to a residual graph include changes in residual capacities and introduction of backwards arcs where needed. The path-finding procedure uses only those changes made to the residual graph that may cause the current augmenting path to be no longer optimal. One such change occurs when the residual capacity of an arc is reduced to zero. This would result from consumption of all remaining capacity of an arc in a given iteration. No flow can be assigned to an arc with no remaining capacity in subsequent iterations unless some capacity is returned to it during some future iteration. Therefore, the current augmenting path from the source to the sink can no longer be used to ship any more supply and a new path must be found. Similarly, introduction of an arc into the residual graph would also create the need to determine whether or not the current augmenting path is the shortest. Arcs may be introduced in the residual graph in two ways: (1) a backwards arc with positive capacity may be added to the network or (2) capacity might be returned to some arc (either real or backwards) with zero residual capacity rendering it available for assigning flow during subsequent iterations.

When an arc's remaining capacity at a particular departure time is reduced to zero, it is said to leave the network for the given departure time and it returns only if capacity is returned. To model this, rather than physically remove the arcs from the residual graph, such changes are represented by changes in arc travel times. An arc with zero remaining capacity will have infinite travel time. When an arc is returned to the residual graph, its travel time will be changed from an infinite value to a finite one. Given this approach, it is possible view the problem of finding the earliest-arrival time-path at each iteration as a problem of updating the previous augmenting path given changes in arc travel times in the network. As a result, a reoptimization technique that can determine an updated augmenting path in the residual graph given changes in arc travel times can be used. Path-finding techniques and a reoptimization procedure for shortest path problem are discussed in Chapter 3.

At each iteration, the Quick-TDQFP heuristic determines a new augmenting path by updating the augmenting path from the prior iteration. The path solutions at each iteration are maintained in the set of distance labels, $\Lambda = \{\lambda_i\}_{i \in N}$, and path pointers,

$\pi = \{\pi_i^1, \pi_i^2\}_{i \in N}$. Λ^p and π^p are used to represent solutions from the prior iteration and form an input to the path-finding algorithm at each iteration. Set Q contains the list of arcs on which the travel times changed when the residual graph was last updated. Each time the path-finding algorithm is called, it returns updated Λ and π given Λ^p, π^p and Q.

Each element of set Q is an arc for which the network attributes changed during the last iteration. Therefore, an arc $(i, j) \in Q$ can be: (1) a backwards arc that did not previously exist and was added to the network in the current iteration, (2) a real arc whose entire capacity at some time t was used up in a previous iteration and some capacity has been returned to it during the current iteration, or (3) an arc, either real or backwards, on which the residual capacity has become zero with the assignment of flow in the current iteration.

2.3 The Quick-TDQFP heuristic – Overview and Steps

In this section, the steps of the Quick-TDQFP heuristic that are different from those of the TDQFP algorithm are discussed.

The Quick-TDQFP heuristic uses the same notation and assumptions as the original TDQFP algorithm. Similar to the TDQFP algorithm, or any other successive shortest path based algorithm, each iteration of the Quick-TDQFP heuristic begins with determining the augmenting path. In the case of Quick-TDQFP heuristic, this augmenting path is determined by REOPT, a reoptimization heuristic that requires a list of travel time updates in the graph (set Q) and the path solution from the prior iteration (Λ^p and π^p). Once the path has been determined, flow is augmented on it. The arcs in set Q are no longer required and new ones will be added to it while updating the residual graph. Therefore, Q is emptied of its contents from the previous iteration. The capacities on the arcs along the chosen path are reduced accordingly. If the residual capacity along any arc becomes zero at some time instance, the arc is added to the set Q and the travel time on the arc is changed to infinity for that time instant. Backwards arcs are added to the residual graph and to the set Q, if they do not already exist. The heuristic stops once all the supply has been shipped from the source.

The steps of the Quick-TDQFP heuristic are the same as that of the TDQFP algorithm. However, the organization of these steps is a little different. Step 1 in the original algorithm has been divided into two steps: Step 1a and Step 1b. The instructions before the residual capacities are updated are covered by Step 1a. Therefore, instructions for calling the path-finding function, augmenting flow along the path and calculation of the remaining supply are executed in Step 1a. Step 1b deals with the calculation of residual capacities and backwards arc updates. That is, the attributes of the network are changed in Step 1b. There is no significant difference in the instructions in the latter part of Step 1 of the original TDQFP algorithm and Step 1b of the Quick-TDQFP heuristic. Only the order in which they are executed is different. This reordering allows easier implementation of the instructions in terms of the Quick-TDQFP heuristic. Only those steps of the Quick-TDQFP heuristic in which travel times are updated and arcs are added to set Q are different from those in Step 1 of the original TDQFP algorithm.

The detailed steps of the Quick-TDQFP heuristic have been provided in Appendix A with an illustrative example. The steps of the Quick-TDQFP heuristic that differ from those of the original TDQFP algorithm, either in order of execution or purpose are as following.

Step 0

Set $Q = \varnothing$ (Additional instruction)

Step 1a

Instead of calling function TDEAT$(k, \hat{t}, T, G(x))$, call function REOPT$(\hat{t}, \Lambda^p, \pi^p, G(x), Q)$, whose output contains $\Lambda, \pi, \alpha_l, \kappa_l,$ and P_l.

Step 1b

Update residual capacities and residual graph:

```
For all $((i,j),t) \in P_l$.
    Set variables $t_i = t$ and $t_j = t + \tau_{ij}(t)$
    $r_{ij}(t_i) = u_{ij}(t_i) - \varepsilon$
    If arc $(j,i) \notin G(x)$
        add arc $(j,i)$ to $G(x)$
        $r_{ji}(t) = \begin{cases} u_{ji}(t) + \varepsilon, t = t_j \\ 0, \forall\{t \in \{0,...T\} \mid t \neq t_j\} \end{cases}$
        $\tau_{ji}(t) = \begin{cases} -\tau_{ij}(t_i), t = t_j \\ \infty, \forall\{t \in \{0,...T\} \mid t \neq t_j\} \end{cases}$
        $Q = Q + \{(j,i)\}$
    Else If arc $(j,i) \in G(x)$ and $u_{ji}(t_j) = 0,$
        $r_{ji}(t_j) = u_{ji}(t_j) + \varepsilon$
```

```
            $\tau_{ji}(t_j) = -\tau_{ij}(t_i)$
            $Q = Q + \{(j,i)\}$
        If $r_{ij}(t_i) = 0$, $Q = Q + \{(i,j)\}$ and set $\tau_{ij}(t_i) = \infty$
    Go to Step 1a.
```

In Step 0, the set *Q* and other variables are initialized. **Step 1a** calls the function REOPT, which is the path-finding function. It needs the prior path solution contained in Λ^p and π^p, the list of arcs on which travel times have been updated (set *Q*), the residual graph, *G*(*x*), and the intended departure time from the source, $\hat{t}$. The intended departure time from the source is the earliest time at which there is some positive amount of supply. The output of the path-finding function is the updated path pointers, π, and distance labels, Λ, and the earliest-arrival-time-path departing from the source at time $\hat{t}$, P_l. The remaining capacity of this path is given by κ_l and the arrival time at the destination by α_l.

Step 1b contains the same instructions as those given in the description of the TDQFP algorithm except where travel times are updated and arcs are added to the set *Q*. Once the flow has been augmented along the path, residual capacities are updated for the arcs in the path. If a corresponding backwards arc does not exist, it is added to the network. This is done by updating its travel time from infinity to a finite value. The arc is then added to set *Q*. If the residual capacity on any arc in the path becomes zero, its travel time is changed to infinity at that time instance and the arc is added to set *Q*.

For example, suppose positive flow was assigned to a real arc (i, j) at time *t*, which means that the flow departs from node *i* at time *t* and arrives at node *j* at time $t' = t + \tau_{ij}(t)$. A backwards arc (j,i) that can be used at time t' will be added to the network with travel time $\tau_{ji}(t') = -\tau_{ij}(t)$ and capacity $r_{ji}(t') = x_{ij}(t)$. The arc (j,i) would be added to the set *Q*. Such backwards arcs permit the return of capacity to an arc in subsequent iterations. If the updated residual capacity on the real arc (i, j) at time *t* is now zero, the arc (i, j) would be added to the set *Q* and $\tau_{ij}(t)$ would be set equal to infinity. If arc (i, j) to which positive flow is assigned at time *t* was a backwards arc, it would result in capacity being returned to the corresponding real arc. If the residual capacity on the corresponding real arc was previously zero, its travel time at $t' = t + \tau_{ij}(t)$ is changed from infinity to its original travel time and the arc (j,i) is added to the set *Q*.

Like the TDQFP algorithm, the Quick-TDQFP heuristic ends with the set of arc flows *x*. If the solution contains positive flow $x_{ji}(t')$ assigned to a backwards arc (j,i) at time $t' = t + \tau_{ij}(t)$, then $x_{ij}(t)$ is reduced by $x_{ji}(t')$.

As discussed, the Quick-TDQFP heuristic relies on REOPT, a heuristic procedure that determines the earliest-arrival-time-path to a given destination departing from a specified source at a fixed departure time. REOPT determines new paths by reoptimizing the augmenting path from the prior iteration given changes in arc travel times. REOPT and other related path-finding procedures are discussed in the next chapter.

Chapter 3: The Path-Finding Algorithms

3.1 Introduction

The Quick-TDQFP heuristic requires a procedure to determine the earliest-arrival-time-path between a specified source and sink in a non-FIFO, time-dependent network with no waiting allowed at any node. This procedure should be able to quickly produce the new augmenting path in the updated residual graph. One approach (the approach taken herein) is to update the augmenting path from the previous iteration when provided with a list of travel time updates to the graph. Such an updated path can be produced by a path reoptimization technique. The TDLTP algorithm (Ziliaskopoulos and Mahmassani, 1993) meets nearly all the requirements for use in the framework of the Quick-TDQFP heuristic. However, it solves a larger problem than is required for solution of a single-source to sink-sink problem. Miller-Hooks and Yang (2004) proposed modifications to the TDLTP algorithm to create its reoptimization version. This version can update paths in a network given multiple changes in arc travel times without resolving the path problem from scratch. The TDLTP algorithm of Ziliaskopoulos and Mahmassani (1993) and the TDLTP Reoptimization algorithm of Miller-Hooks and Yang (2004) are reviewed in Section 3.2. A limitation of their reoptimization approach for application to the Quick-TDQFP heuristic is discussed in Section 3.3 and a heuristic technique to adapt the TDLTP Reoptimization algorithm to deal with the specific limitation encountered in this context is presented in Section 3.4. In Section 3.5 the Chrono-SPT algorithm of Pallottino and Scutella (1998) is discussed. A modified implementation of the Chrono-SPT algorithm that enables the algorithm to work with negative arc travel times is also proposed in Section 3.4**.** This implementation is called the M-Chrono-SPT algorithm.

3.2 The TDLTP and the TDLTP Reoptimization Algorithms

The TDLTP algorithm is a label-correcting algorithm that solves for the time-dependent least time paths from all nodes to a given destination for all departure times in a discretized non-stationary time period of interest. Though Ziliaskopoulos and Mahmassani (1993) assumed the arc travel times to be non-negative real values, the algorithm can produce optimal results when considering negative arc travel times. Miller-Hooks and Yang (2004) proposed modifications to the TDLTP Reoptimization algorithm to create its reoptimization version.

The TDLTP Reoptimization algorithm begins with optimal distance labels for each node, referred to as the "optimal node potential" from the solution to the previous problem instance. In this thesis, the term "label" instead of "node potential" is used throughout. When provided with the updated travel times and the set of arcs on which the travel times have been updated, the algorithm iteratively updates the previous labels until they are optimal. In addition to the node labels, the algorithm employs two path pointers which indicate the next node on the path to the destination and the arrival time at that node to define the solution. Each time a label is updated, the corresponding path pointers are updated, as well.

Initially, the end nodes of arcs on which travel times have been updated are entered into a FIFO scan eligible (SE) list. In the Quick-TDQFP heuristic this list of arcs on which travel times have been updated is maintained in the set *Q*. At each iteration, a node (called the current node) is removed from this list and the labels of all its predecessor nodes are considered for updating. For each predecessor node, at each time interval, a temporary label is computed using the labels at the current node. The labels at the current node are considered at possible arrival times at the current node using the arc connecting the two nodes. If the value of the temporary label is lower than that of the current label at the predecessor node, the label is updated to the value of the temporary label and the path pointers are set accordingly. In case the value of the temporary label is greater than the current label and the path pointer pointed to the current node, temporary labels are computed using all the successor nodes of the predecessor node under consideration. The label at the predecessor node is then set to the minimum among the temporary labels and the path pointers are updated accordingly.

Miller-Hooks and Yang (2004) proposed the TDLTP Reoptimization algorithm only for positive real-valued arc travel times. However, in a time-dependent residual graph, backwards arcs have negative arc travel times. This poses the requirement of an algorithmic paradigm that can update time-dependent shortest paths in the presence of negative arc travel times. Since the original TDLTP algorithm can produce optimal results when a graph contains arcs with negative travel times, it would seem feasible for the TDLTP Reoptimization algorithm to be able to do the same. Indeed, the TDLTP Reoptimization algorithm can work optimally in the presence of arcs with negative travel times, except when the graph has zero-sum cycles.

3.3 Shortest-Path Algorithms and Zero-Sum Cycles

At each iteration of the Quick-TDQFP heuristic, the time-dependent residual network is updated by the addition of backwards arcs (or the travel time of the arcs is updated from infinity to a finite negative value) and calculation of residual capacities. The travel time on a backwards arc at the time at which it can be used is equal to the negative of that of the corresponding real arc at the time at which it was used. For example, suppose positive flow was assigned to a real arc (i, j) at time *t*, which means that the flow departed from node *i* at time *t* and arrived at node *j* at time $t' = t + \tau_{ij}(t)$. When updating the residual network, a backwards arc (j, i) that can be used at time t' will be added to the network with travel time $\tau_{ji}(t') = -\tau_{ij}(t)$. The arcs (i, j) at time *t* and (j, i) at time t' form a two-node zero-sum cycle since the sum of their weights is zero. Such a two node cycle is defined as an immediate zero-sum cycle. This is shown in Figure 1.

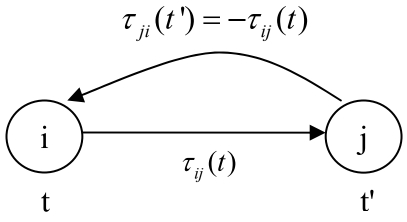


Figure 1: Immediate zero-sum cycle

Note that in a residual graph every real arc to which some flow is assigned without using its full capacity will be a part of an immediate zero-sum cycle composed of the arc and its corresponding backwards arc.

In non-FIFO time-dependent residual graphs, it is possible to have zero-sum cycles composed of more than two nodes (or arcs). Zero-sum cycles in residual graphs composed of more than two nodes are defined as extended zero-sum cycles. Figure 2 shows an extended zero-sum cycle. In the figure, the arc travel times are shown next to the arcs with the time instant at which this travel time is encountered given parenthetically. Consider that at t=0 some amount of flow was shipped along arc (*i,k*) (not shown in the figure) that required a travel time of 5 units. This flow would arrive at node k at time $t=5$. Therefore, arc (*k,i*) with a travel time of -5 at time $t=5$ would be introduced into the network. The extended zero-sum cycle that will be created in the graph with the introduction of the backwards arc can be seen in Figure 2.

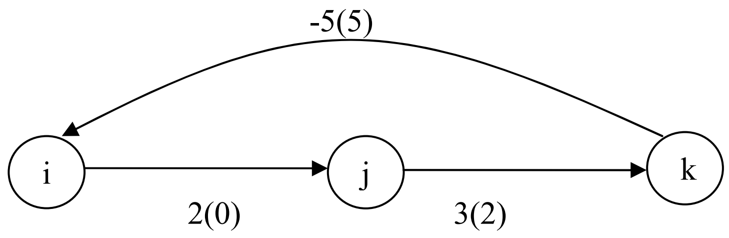


Figure 2: Extended zero-sum cycle

It is apparent that in the successive shortest path framework one will always have a significant number of zero-sum cycles and at each iteration this number may grow.

In time-dependent networks, a zero-sum cycle consists of nodes at specific times, i.e, node-time pairs. The only way it would be possible to return to a node-time pair that was visited earlier in the same path would be to go around a zero-sum cycle. For example, see Figure 2. Assume that the departure time from node i is zero. Flow can get back to node i at time zero by going around the zero-sum cycle where it would depart from node j at time $t=2$ to arrive at node k at $t=5$. By departing from node k at t=5 and using arc *(k,i)* the flow can reach node i at t=0. Thus, in time-dependent graphs, zero-sum cycles may be defined as those cycles that allow the return of flow to a node at the same departure time, no matter how many times the cycle is traversed.

Note that in a time-expanded network, the only possible cycle is a zero-sum cycle. Furthermore, it is well known that in a non-FIFO time-dependent network with no-waiting allowed at any node, shortest path solutions may contain cycles, because it might be possible to depart later from a node and arrive at the destination earlier along the same path. Since waiting at an intermediate node is not permitted, it is sometimes advantageous to cycle until conditions improve. Such cycles do not visit the same node at the same time; rather they arrive at the node at later and later times. Thus, unlike a zero-sum cycle, in a time-expanded graph, a positive cost cycle is merely a directed path between node-time pairs. The inclusion of positive cost cycles in the solution does not pose a problem. However, the same is not true of a zero-sum cycle.

Consider the example shown in Figure 3. The travel times on each arc are shown for a specific time instant given parenthetically next to the travel time. For example, arc (1,4) has a travel time of 1 at time *t=0* and arc (4,5) has a travel time of 1 at time *t=1*. Suppose node 5 is the destination node. We are given the following all-to-one solution in terms of path pointers $\pi_i^1(t)$ and $\pi_i^2(t)$: $\pi_1^1(0)=2$, $\pi_1^2(0)=1$; $\pi_2^1(1)=3$, $\pi_2^2(1)=2$; $\pi_3^1(2)=1$, $\pi_3^2(2)=0$; $\pi_4^1(1)=5$ and $\pi_4^2(1)=2$, where $\pi_i^1(t)$ points to the successor node and $\pi_i^2(t)$ gives the arrival time at the successor node.

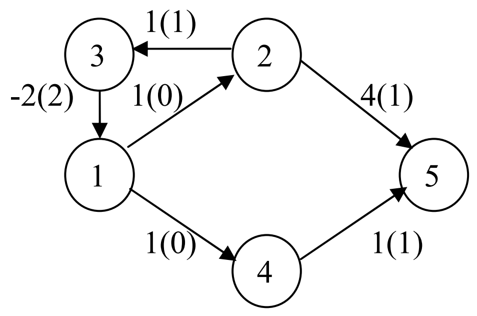


Figure 3: Graph for illustrative example

If an attempt to construct the paths based on this solution were to be made, one would get stuck in a non-terminating cycle. Starting with node 1 at *t=0*, one would arrive at node 2 at *t=1*, from where one would reach node 3 at *t=2*. On departure from node 3 at *t=2*, one would end at node 1 at *t=0* and this cycle would continue. Even if it was somehow possible to construct the paths based on this solution, they would look like those shown in Figure 4.

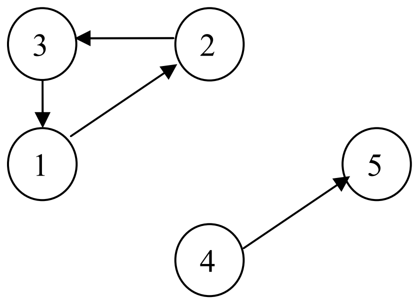
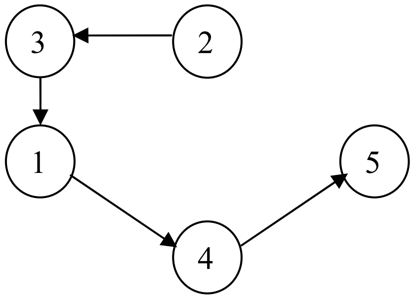


Figure 4: Solution with a zero-sum cycle.

Clearly, the presence of a zero-sum cycle in the solution of a path-finding algorithm is a problem. If the cycle formed by nodes 1, 2 and 3 were not a zero-sum cycle, the path pointer $\pi_3^2(2)$ would not have been equal to zero, and thus even though node 1 would have been on the path again, the arrival time at node 1 would be different, and thus the path followed from there would have been different. Such a path would eventually have terminated at the destination node at some time unless there was a zero-sum cycle further in the path.

The TDLTP Reoptimization algorithm, which was not designed to deal with negative arc travel times, does not guarantee its solution to be free of zero-sum cycles. The example shown in Figure 3 can be used to illustrate where the TDLTP Reoptimization algorithm fails. Assume that the travel times given on the arcs are time-invariant, i.e., the arc has the same travel time no matter what time it is used. The optimal all-to-one shortest path solution tree for the example is shown in Figure 5.

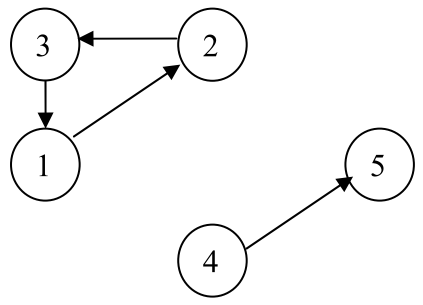


$$\begin{aligned}
&\lambda_1(1) = 2, \pi_1^1(1) = 4, \pi_1^2(1) = 2\\
&\lambda_2(2) = 1, \pi_2^1(2) = 3, \pi_2^2(2) = 3\\
&\lambda_3(3) = 0, \pi_3^1(3) = 1, \pi_3^1(3) = 1\\
&\lambda_4(2) = 1, \pi_4^1(2) = 5, \pi_4^1(2) = 3\\
&\lambda_5(t) = 0
\end{aligned}$$

Figure 5: Shortest Path Tree For Departure from Node 1 at *t=1*

The distance label $\lambda_i(t)$ and the path pointers $\pi_i^1(t)$ and $\pi_i^2(t)$ for each node *i* are indicated next to the tree. Suppose that the travel time on arc (1,4) is updated to 6 at time *t=1*. Using the TDLTP Reoptimization algorithm, node 4 is added to the SE list. In the first iteration, the label at node 1 at time *t=1* is considered for updating and the temporary label η_1 is computed as $\eta_1 = \lambda_4 + 6 = 7$. Since η_1 > $\lambda_1(1)$ and $\pi_1^1(1) = 4$, step 2b of the TDLTP Reoptimization algorithm is invoked and the label $\lambda_1(1)$ would be set equal to the minimum of the label obtained by considering all the successor nodes of node 1. Thus

$\lambda_1(1) = 2$ with $\pi_1^1(1) = 2$ and $\pi_1^2(1) = 2$. Node 1 is added to the SE list. When node 1 is scanned, the label at node 3 at *t=3*, $\lambda_3(3)$, is considered for updating, but since the temporary label and the current label are equal nothing is updated. The algorithm terminates because the SE list is empty. The resulting shortest path tree that can be constructed from the path pointers is shown in Figure 6.



$$\begin{aligned}
&\lambda_1(1) = 2, \pi_1^1(1) = 2, \pi_1^2(1) = 2 \\
&\lambda_2(2) = 1, \pi_2^1(2) = 3, \pi_2^2(2) = 3 \\
&\lambda_3(3) = 0, \pi_3^1(3) = 1, \pi_3^1(3) = 1 \\
&\lambda_4(2) = 1, \pi_4^1(2) = 5, \pi_4^1(2) = 3 \\
&\lambda_5(t) = 0
\end{aligned}$$

Figure 6: TDLTP Reoptimization algorithm solution

Clearly, an added condition to guarantee that the end solution of the algorithm would be a spanning tree, and can produce near-optimal (if not optimal) distance labels is required. Such a condition is presented in the form of a heuristic procedure in the next section.

3.4 The Path Diving Procedure

In the TDLTP Reoptimization algorithm, updating a path pointer implies changing an arc in the solution tree. In the presence of zero-sum cycles in the graph, the inclusion of some arcs in the solution tree might cause the solution to have a zero-sum cycle. Thus, solutions with zero-sum cycles could be avoided if, before including an arc in the solution, one could check whether it would create a zero-sum cycle. Algorithmically, this may be achieved by checking the existing path from a node to the destination, before updating any labels using that path. For instance, suppose the distance label at node *i* at time *t*, $\lambda_i(t)$, is to be considered for updating. It will be updated using a node $j \in \Gamma^+(i)$, where $\Gamma^+(i)$ is the set composed of all the successor nodes of node *i*. Thus, if the existing path from any node $j_{cycle} \in \Gamma^+(i)$ at time $t + \tau_{ij_{cycle}}(t)$ leads one back to node *i* at time *t*, updating $\lambda_i(t)$ using $\lambda_{j_{cycle}}(t + \tau_{ij_{cycle}}(t))$ (i.e., setting $\pi_i^1(t) = j$ and $\pi_i^2(t) = t + \tau_{ij_{cycle}}(\text{t})$)[5] would create a zero-sum cycle in the solution. Therefore, node $j_{cycle} \in \Gamma^+(i)$ should be excluded from the set of nodes considered for updating the label $\lambda_i(t)$. Such a node would be referred to as a 'tainted' node in the remainder of this thesis. A heuristic procedure called PathDive is introduced in the TDLTP Reoptimization algorithm that achieves the goal of ensuring that the solution has no zero-sum cycles. While, PathDive guarantees optimal results in the presence of immediate zero-sum cycles, the same is not true in the presence of extended zero-sum cycles.

Specifically, the path diving heuristic is employed prior to computing a temporary label in the TDLTP Reoptimization algorithm. This computation occurs at the beginning of step 2, when labels at each predecessor node of the scanned node are considered for updating, and in step 2b when labels at all successor nodes of the concerned predecessor node are considered in updating the label at the predecessor node (See Miller-Hooks and

[5] The same notation as used by Miller-Hooks and Yang (2004) is used for denoting path pointers.

Yang, 2004 for details). Step 2 of TDLTP Reoptimization algorithm with the Path Diving heuristic is given next.[6] Note that $\Gamma^{-}(i)$ refers to the set of predecessor nodes of node *i*.

```
Step 2
For each predecessor node i of node j, i.e. $\forall i \in \Gamma^{-1}(j)$,
For each $t \in S$,
   if $PathDive(i, j, t) =$ false
      compute the temporary label $\eta_i(t) = \tau_{ij}(t) + \lambda_j(t + \tau_{ij}(t))$.
      if $\eta_i(t) < \lambda_i(t)$, then
        Step 2a. $\lambda_i(t) = \eta_i(t), \pi_i^1(t) = j, \pi_i^2(t) = t + \tau_{ij}(t)$ and SE = SE $\cup$ {i}
     else
       Step 2b. if $\eta_i(t) > \lambda_i(t)$ and $\pi_i^1(t) = j$
         for every successor node k of node i including node j, i.e.$\forall v \in \Gamma^{+}(i)$
              if $PathDive(i, v, t) =$ false
                  compute $\eta_i^v(t) = \tau_{iv}(t) + \lambda_v(t + \tau_{iv}(t))$
         $\lambda_i(t) = \min_{v \in \Gamma^{+}(i): PathDive(i,v,t) = false} (\eta_i^v(t))$,
         $v' = \arg\min_{v \in \Gamma^{+}(i): PathDive(i,k,t) = false} (\eta_i^v(t)), \pi_i^1(t) = v', \pi_i^2(t) = t + \tau_{iv'}(t)$
         and SE = SE $\cup$ {i}
```

The function PathDive is as following.

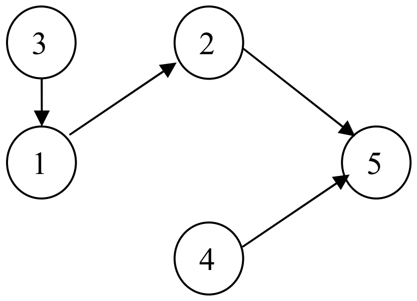
```
$PathDive(i, j, t)$
$\beta_1 = \pi_j^1(t + \tau_{ij}(t)), \beta_2 = \pi_j^2(t + \tau_{ij}(t))$
$while(\beta_1 \neq$ sink node)
   $\beta_1 = \pi_{\beta_1}^1(\beta_2)$ and $\beta_2 = \pi_{\beta_1}^2(\beta_2)$
   if $\beta_1 = i$ and $\beta_2 = t$, return 1 and Stop $PathDive$
Return 0
```

The function PathDive tracks the path from the given successor node *j* of node *i* by using the path pointers until it reaches the sink node. If it encounters node *i* at time *t* on this

[6] The same notation as used in Miller-Hooks and Yang (2004) is used.

path, it returns a value of 1 (true) indicating that including arc (*i,j*) at time *t* in the solution would cause a zero-sum cycle in the solution. That is, node *j* is a tainted node. The TDLTP Reoptimization algorithm computes the temporary label values considering only those successor nodes that are not tainted, i.e. for which the PathDive function returns a value of 0 (false). Thus, it ends with no zero-sum cycles in the solution.

The TDLTP Reoptimization algorithm with path diving heuristic is illustrated on the example given in Figure 3. Unlike when the original TDLTP Reoptimization algorithm is used, in the first iteration, $\lambda_1(1)$ cannot be updated using node 2 during the execution of Step 2b of the algorithm, because the function PathDive will return a value of 1 indicating that node 2 is a tainted node. Thus, in the first iteration $\lambda_1(1)$ = $\lambda_4 + 6 = 7$ and $\pi_1^1(1) = 4$. Node 1 is added to the SE list. When node 1 is scanned, the label for node 3, $\lambda_3(3)$, is updated to $\lambda_1(1) + \tau_{31} = 7 - 2 = 5$ with $\pi_3^1(3) = 1$ and node 3 enters the SE list. On scanning node 3, a temporary label is computed for node 2, $\eta_2 = \lambda_3(3) + 1 = 6 > \lambda_2(2)$. Since $\pi_2^1(2) = 3$, Step 2b of the algorithm is executed. The function PathDive returns a value of 0 for each of the successor nodes of node 2, and $\lambda_2(2)$ the label at node 2 is set to $\eta_2^5 = \lambda_5(6) + 4 = 4$ with $\pi_2^1(2) = 5$ and node 2 enters the SE list. When node 2 is scanned, $\lambda_1(1)$ is updated to 5 with $\pi_1^1(1) = 2$. In the subsequent iteration $\lambda_3(3)$ is updated accordingly. The final solution is given in Figure 7. Incidentally, this is the optimal solution.



$$\lambda_1(1) = 5, \pi_1^1(1) = 2, \pi_1^2(1) = 2$$
$$\lambda_2(2) = 4, \pi_2^1(2) = 5, \pi_2^2(2) = 6$$
$$\lambda_3(3) = 3, \pi_3^1(3) = 1, \pi_3^1(3) = 1$$
$$\lambda_4(2) = 1, \pi_4^1(2) = 5, \pi_4^1(2) = 3$$
$$\lambda_5(t) = 0$$

Figure 7: Solution using PathDive Heuristic

Computational Complexity and Validity of the Heuristic

The TDLTP Reoptimization algorithm starts with a solution to the prior path problem instance, which is a set of shortest path trees in the time-expanded network. The TDLTP Reoptimization algorithm with the path diving heuristic only performs operations where arcs in the tree are swapped with those not in the tree. Thus, the final solution has the same number of arcs as the original solution. In a time-expanded graph (TDLTP works on an implicit time expansion), the only possible cycle is a zero sum-cycle. Every time a label is updated by the algorithm the path diving function is used to make sure that no zero-sum cycles are present in the solution.Thus, the heuristic terminates with a solution that has no zero-sum cycles. Therefore, the solution has the same number of arcs as the previous solution and no zero-sum cycles (or cycles in the time-expanded network).

Note that arc replacements are made only in Step 2b of the algorithm. In Step 2a, only the label is updated. Each arc replacement in Step 2b restores connectivity by connecting the node whose label is under consideration for updating to either the same node as in the previous solution tree but at a later arrival time, or to another successor node at an appropriate arrival time. This means that connectivity is maintained, there are no cycles (in the time-expanded graph) and the total number of arcs has not changed. Thus, the original structure, i.e. a minimum spanning tree rooted at the destination at each time instance, is retained.

However, the path diving heuristic is not always guaranteed to produce an optimal solution in terms of distance labels. In the example network in Figure 3, the network topology was such that the heuristic terminated with the optimal solution. However, consider the following network in Figure 8, which is a modification of the network in Figure 3. Node 6 and three arcs, (1,6), (6,5) and (3,5), have been added to the network in Figure 3. Consider the time-invariant travel time on arc: (3,5) to be 0; (1,6) to be 3; and (6,5) to be 1. The travel time on the rest of the arcs is the same as shown in Figure 3 and the destination node is node 5.

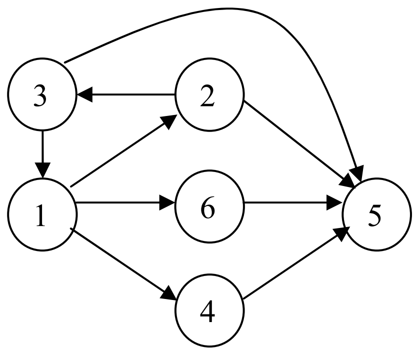
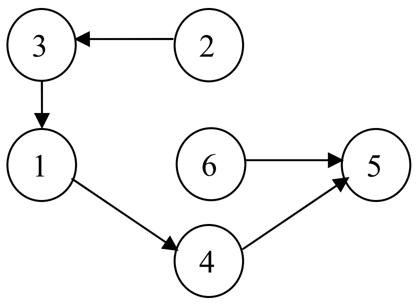


Figure 8: Network for Counter-Example

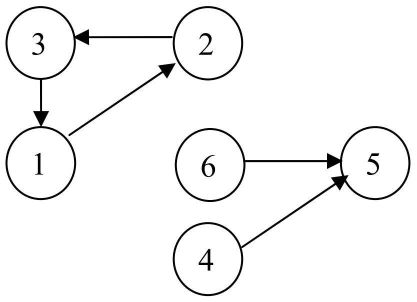
One possible optimal solution tree to the problem without any travel time updates is shown in Figure 9.



$$\lambda_1(1) = 2, \pi_1^1(1) = 4, \pi_1^2(1) = 2$$
$$\lambda_2(2) = 1, \pi_2^1(2) = 3, \pi_2^2(2) = 3$$
$$\lambda_3(3) = 0, \pi_3^1(3) = 1, \pi_3^1(3) = 1$$
$$\lambda_4(2) = 1, \pi_4^1(2) = 5, \pi_4^1(2) = 3$$
$$\lambda_6(2) = 1, \pi_6^1(2) = 5, \pi_6^1(2) = 3$$
$$\lambda_5(t) = 0$$

Figure 9: Shortest Path Tree For Departure from Node 1 at *t=1*

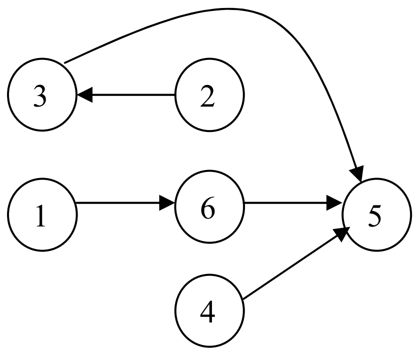
The solution using the original TDLTP Reoptimization algorithm when the travel time on arc (1,4) is updated to 6 at time *t=1* is given in Figure 10.



$$\lambda_1(1) = 2, \pi_1^1(1) = 2, \pi_1^2(1) = 2$$
$$\lambda_2(2) = 1, \pi_2^1(2) = 3, \pi_2^2(2) = 3$$
$$\lambda_3(3) = 0, \pi_3^1(3) = 1, \pi_3^1(3) = 1$$
$$\lambda_4(2) = 1, \pi_4^1(2) = 5, \pi_4^1(2) = 3$$
$$\lambda_6(2) = 1, \pi_6^1(2) = 5, \pi_6^1(2) = 3$$
$$\lambda_5(t) = 0$$

Figure 10: TDLTP Reoptimization algorithm solution

The application of the path diving heuristic to the example network from Figure 8, given a travel time update on arc (1,4) at time *t=1* and the solution shown in Figure 9 results in steps similar to those with the network in Figure 3. That is, in the first iteration, $\lambda_1(1)$ cannot be updated using node 2 during the execution of Step 2b of the algorithm, because the function PathDive will return a value of 1. The label derived using the label at node 6 is the minimum possible. Thus, $\lambda_1(1)$ = $\lambda_6(4) + 3 = 4$ and $\pi_1^1(1) = 6$. Node 1 enters the SE list and the label at node 3 is considered for updating. Since, the computed temporary label, $\eta = \lambda_1(1) + \tau_{31}(3) = 2$ is greater than $\lambda_3(3)$ and $\pi_3^1(3) = 1$, step 2b of the heuristic is invoked. Successor nodes of node 3, i.e., nodes 1 and 5 are considered in updating its label. The label derived by using node 5 is the minimum of the two. Thus, $\lambda_3(3) = 0$ and $\pi_3^1(3) = 5$. When node 3 is scanned, the corresponding label at node 2 does not change since, the temporary label and the present label at node 2 are equal. The heuristic terminates with the solution shown in Figure 11.



$\lambda_1(1) = 4, \pi_1^1(1) = 6, \pi_1^2(1) = 4$
$\lambda_2(2) = 1, \pi_2^1(2) = 3, \pi_2^2(2) = 3$
$\lambda_3(3) = 0, \pi_3^1(3) = 5, \pi_3^1(3) = 3$
$\lambda_4(4) = 1, \pi_4^1(4) = 5, \pi_4^1(4) = 5$
$\lambda_6(4) = 1, \pi_6^1(2) = 5, \pi_6^1(4) = 5$
$\lambda_5(t) = 0$

Figure 11: Sub-Optimal Solution with PathDive

The label at node 1, $\lambda_1(1)$, is not optimal in this solution. Its value can be improved by using node 2 as the successor node. In that case the $\lambda_1(1)$ would be equal to 1 as opposed to its present value of 4. The label, $\lambda_1(1)$, was not updated to the optimal value because the label at node 2 did not have to be updated and thus node 2 never entered the SE list. Note that node 2 is what was defined as a tainted node. In light of this example, it can be concluded that the TDLTP Reoptimization algorithm with the PathDive function is not guaranteed to produce an optimal solution in the presence of an extended zero-sum cycle when the label for at least one of the constituent nodes of the cycle except the tainted node can be retained at its current value by changing the successor node its path pointer points to. In the counter-example provided prior to the conclusion, the value of the label at node 3 does not change, though its path pointer changes. Thus, the label at the tainted node, node 2, is never considered for updating and it never enters the SE list. Thus node 2 is never scanned. In other words, if the tainted node is not eventually considered in updating the label at the current node, the heuristic might terminate with a sub-optimal solution.

It must be noted that in the above example, once the label $\lambda_1(1)$ had been computed, had any of its successor nodes entered the SE list, the labels at node 1 would have been considered for updating again. In which case, the label at node 1 might have been updated to the optimal value. In bigger networks, with more number of time intervals and a higher in and out-degree, the possibility of one of the successor nodes of node 1 entering the SE list would be quite high. Thus, the probability that the label at node 1 might get updated to the optimal value would be quite high. In the numerical experiments reported in Chapter 4, it was found that the heuristic always ended with an optimal solution.

Unlike the TDLTP Reoptimization algorithm, the heuristic is guaranteed to terminate without any zero-sum cycles in the solution. Given that the network topology is such that the condition stated in the preceding paragraph where the heuristic fails to produce an optimal solution does not occur, it can be seen that the TDLTP Reoptimization algorithm with PathDive function will produce optimal results.

Suppose that the label at node *i* at time *t*, $\lambda_i(t)$, is being considered for an update. The only difference between the original TDLTP Reoptimization algorithm without and with the PathDive function is that when PathDive is included, certain successor nodes

$j \in \Gamma^{+}(i)$ are not considered in updating the labels at node *i* in an iteration. A successor node $j_{cycle} \in \Gamma^{+}(i)$ is not considered in updating labels at node *i* if and only if it is a tainted node, i.e. its inclusion in the solution would cause the solution to have a zero-sum cycle. Thus, currently, the path from node j_{cycle} at time $(t + \tau_{ij_{cycle}}(t))$ to the destination node passes through node *i* at time *t*. This means the label $\lambda_{j_{cycle}}(t + \tau_{ij}(t))$ is a function of $\lambda_i(t)$. Given that the labels for none of the constituent nodes of the zero-sum cycle can be retained at their present value, if $\lambda_i(t)$ is updated to some new value all the nodes in the zero-sum cycle would enter the SE list. Thus, node j_{cycle} would enter the SE list, after the label $\lambda_i(t)$ had been updated. The label $\lambda_i(t)$ would therefore be considered for updating again when the node j_{cycle} is scanned since $i \in \Gamma^{-}(j_{cycle})$. This implies that the TDLTP reoptimization algorithm would terminate with optimal distance labels given that the above mentioned condtion holds.

The worst-case computational complexity of the TDLTP Reoptimization algorithm with the PathDive function is $\sim O(m \cdot n^2 \cdot I^2)$, where *m* is the number of arcs in the graph, *n* is the number of nodes, and *I* is the number of discrete time intervals in the period of interest. The path diving procedure is an $O(n)$ step (in the worst case the path from a node to the destination node will contain n nodes) and it is called each time the predecessor nodes are examined. Miller-Hooks and Yang (2004) show that for the TDLTP Reoptimization algorithm, the maximum number of times predecessor nodes are examined is $m \cdot I$ and that the complexity of the examination is $\sim O(n \cdot I)$. Since, an additional $O(n)$ path diving check is now performed during each examination, the complexity of examination becomes $\sim O((n \cdot I) \cdot (n))$ or $O(n^2 \cdot I)$. Therefore, the overall complexity of the TDLTP Reoptimization algorithm with the path diving heuristic is $O((n^2 \cdot I) \cdot (m \cdot I))$ or $\sim O(m \cdot n^2 \cdot I^2)$.

3.5 The M-Chrono SPT algorithm

In time-dependent, non-FIFO networks it is known that better solutions can sometimes be obtained by waiting at certain nodes. In this section, a quick and exact algorithm for path-finding is presented that allows waiting only at the source node and at no other node in the network. The algorithm is called the M-Chrono-SPT algorithm (Multiple-Chrono-SPT) and can be seen as a variation on the Chrono-SPT algorithm of Pallottino and Scutella (1998). Unlike the TDLTP Reoptimization algorithm with the path diving heuristic procuedure, the M-Chrono-SPT algorithm terminates with optimal shortest path solutions. The M-Chrono-SPT is quicker than the TDLTP algorithm because unlike the latter, the M-Chrono-SPT algorithm does not determine labels for each node at each time interval. Instead, it only determines labels for the nodes at those times at which it is possible to reach those nodes when departing from the source at some time within the time period of interest. This reduces the required computational effort as compared to that needed to solve the all-to-one shortest path problem for all departure times (as solved by TDLTP algorithm).

The Chrono-SPT algorithm of Pallottino and Scutella (1998) determines the shortest paths from a given source to all other nodes in the network for each arrival time at each node in a time-dependent and non-FIFO environment. The algorithm operates on a time-expanded network where the time period of interest has been discretized into small intervals and it requires that arc weights are positive. The M-Chrono-SPT algorithm modifies the Chrono-SPT algorithm to produce optimal results for a graph with non-positive arc travel times.

In the Chrono-SPT algorithm, a label is associated with each node at each time interval. Each label maintains the total travel time on the shortest path from the source to the associated node for each arrival time at that node. These labels are initialized to infinity, providing an upper bound on the solutions. The label values are improved (reduced) during the course of the algorithm as lower cost solution paths are determined. An associated path pointers point to the predecessor node on the path and the departure time from that node. There is a "bucket" associated with each time interval. These buckets initially contain only the source node. The algorithm begins at the first time interval and proceeds through each time interval in chronological order. At each time *t* the nodes from the bucket are removed and scanned. For each successor node of a node that is scanned, a temporary label is calculated at the time one would arrive at the successor node by departing from the scanned node at time *t*. Whenever the value of the temporary label is lower than the value of the current label at the successor node, the label is set to the temporary label and the successor node is added to the bucket at that arrival time. The algorithm terminates once the bucket for the last time interval is empty.

The Chrono-SPT algorithm terminates with labels for each node at each arrival time. The labels indicate the least possible travel time between the source and that node for each arrival time. The algorithm determines the optimal departure time from the source to arrive at each node at the earliest possible time by incurring the least possible travel time.

The Chrono-SPT algorithm works only for nonnegative arc travel times. If any of the travel times are negative, a node might be inserted into a bucket at a time instance associated with an already emptied bucket. Even when the bucket for the last time interval is empty (which is the termination criteria for Chrono-SPT algorithm) there could possibly be a non-empty bucket at some earlier time interval. If the algorithm could revisit a bucket at an earlier time instance, it could possibly deal with negative arc travel times. The M-Chrono-SPT algorithm is based on this notion and can produce optimal solutions considering possible negative arc travel times. If during any iteration a label is updated at a time instance earlier than the departure time from the node under consideration (i.e., the time instance associated with the bucket from which the nodes are currently being removed), the algorithm will revisit all the buckets beginning from the one associated with the first time interval. Thus, the M-Chrono-SPT algorithm goes through multiple cycles of the Chrono-SPT algorithm until all the buckets are empty. Specifically, each time a label is updated using a negative travel time arc, the M-Chrono-SPT algorithm repeats the process of visiting all buckets beginning with the one associated with the first time interval.

The steps of the M-Chrono-SPT algorithm are as following.

Algorithm M-Chrono-SPT

Step 0

Initialize:

$\lambda_i(t) = \infty, \forall i \in N \setminus \{k\}, \forall t \in \{0,...,T\}$

$\pi_i^1(t) = 0, \forall i \in N \setminus \{k\}, \forall t \in \{0,...,T\}$

$\pi_i^2(t) = -1, \forall i \in N \setminus \{k\}, \forall t \in \{0,...,T\}$

$\lambda_{i=k}(t) = 0, \forall t \in \{0,...,T\}$

$\pi_i^1(t) = k, \forall t \in \{0,...,T\}$
$\pi_i^2(t) = -1, \forall t \in \{0,...,T\}$

$SE(t) = SE(t) + \{k\}, \forall t \in \{0,...,T\}$

Step 1

$rpt = 0$

Step 2

```
For $t = 0$ through $t = T$
        Select node $i \in SE(t)$
                For each successor node $j$ of node $i$
                        $\eta_j(t+\tau_{ij}(t)) = \lambda_i(t) + \tau_{ij}(t)$
                        if $\eta_j(t+\tau_{ij}(t)) < \lambda_j(t+\tau_{ij}(t))$
                                $\lambda_j(t+\tau_{ij}(t)) = \eta_j(t+\tau_{ij}(t))$
                                $\pi_j^1(t+\tau_{ij}(t)) = i$ and $\pi_j^2(t+\tau_{ij}(t)) = t$
                                $SE(t+\tau_{ij}(t)) = SE(t+\tau_{ij}(t)) + \{j\}$
                                $if\,(t+\tau_{ij}(t)) < t,$ set rpt = 1
        If $SE(t)$ is empty and $t \neq T$, $t = t+1,$ continue **Step 2**
        Else if $SE(t)$ is empty and $t = T$ and rpt $= 1$, set $t = 0$ go to **Step 1**
            Else
                Go to **Step 3**
```

Step 3 Stop.

Step 0 is simply an initialization step. In Step 1, rpt, the binary indicator variable that indicates whether one needs to cycle through all buckets again is set to zero. If during the course of Step 2, any label is updated using an arc with a negative travel time arc, rpt will be set to 1, indicating that all the buckets need to be visited again. Step 2 is simply the Chrono-SPT algorithm of Pallottino and Scutella (1998).

Algorithm correctness and computational complexity

It can be verified that Step 2 is an implementation of the Chrono-SPT algorithm presented in Pallottino and Scutella (1998) in a non time-expanded graph. Each $SE(t)$ is the equivalent of a bucket in the Chrono-SPT algorithm. As in the Chrono-SPT algorithm, a node enters $SE(t)$, if its label has been improved at that arrival time. Since a typical iteration of the M-chrono-SPT algorithm is the same as the Chrono-SPT algorithm and in both the algorithms a node enters a bucket if and only if its label is changed, as long as all the buckets are empty when the algorithm terminates, the solutions obtained from the M-Chrono-SPT algorithm are optimal.

Suppose at some time t_i an arc with a negative arc travel time is used to update the label for node j at time t_j, such that $t_j < t_i$. The node j is entered into SE(t_j) and the variable *rpt* is set to 1. This forces the algorithm to revisit the bucket at time t_j and pull out node j at the time and reconsider labels for other nodes based on the update in the label of node j at time t_j. The node j is pulled out of the bucket at time t_j. Therefore, the M-Chrono-SPT algorithm does not terminate with a non-empty bucket even when using negative travel times as long as there are no negative cycles.

It remains to be shown that the M-Chrono-SPT algorithm will cycle through the buckets only a finite number of times. The algorithm cycles through all buckets more than once if during the first cycle at least one label is updated using a negative arc. Suppose that the algorithm cycles through all buckets an infinite number of times. This can happen only if at least one label is updated using an arc with a negative travel time an infinite number of times. Since, there are no negative cycles, and only a finite number of arcs with negative travel times, this means at least one of them has to be used an infinite number of times. Thus, it would require that it is possible to reach a particular node at a particular time an infinite number of times and be possible to reduce the label at a successor node each time. Since this is not possible unless there is a negative cycle, if the graph does not have negative cycles (as is the case here), the M-Chrono-SPT algorithm would cycle through all buckets only a finite number of times.

Pallottino and Scutella (1998) show that the computational complexity of the Chrono-SPT paradigm in a non-FIFO network with no-waiting allowed is $O(m \cdot I)$, where m is the number of arcs in the graph, and I is the number of discrete time intervals in the period of interest.. The M-Chrono-SPT algorithm revisits all buckets starting from the one associated with the first time interval if a label is updated using an arc with a negative travel time during an iteration. It may be noted that the algorithm starts revisiting buckets only after it has reached the bucket associated with the last time interval. In this light, one may see M-Chrono-SPT algorithm as multiple calls to the Chrono-SPT paradigm. Since, a call is made only when a label is updated using an arc with a negative travel time, the maximum number of times the Chrono-SPT algorithm can be called is $m \cdot I$ times. Therefore, the computational complexity of the M-Chrono-SPT algorithm is $O((m \cdot I) \cdot (m \cdot I))$ or $O(m^2 \cdot I^2)$.

In the next chapter, results from the computational experiments are reported and analyzed.

Chapter 4: Computational Results

4.1 Introduction

In this chapter, the performance, in terms of average run time, of the Quick-TDQFP heuristic is compared with that of the original TDQFP algorithm through numerical experiments on several randomly generated networks. The implementation of the TDQFP algorithm employed in the experiments relies on the TDLTP algorithm as its path-finding procedure. Run times for the TDQFP algorithm with source-only waiting (using M-Chrono-SPT algorithm for path-finding) are also presented. The algorithms were programmed in C++ and run on a DEC Alpha XP1000 professional workstation with 1 gigabyte ram and 2 gigabyte swap, running Digital 4.0E operating system, using Digital's C++ compiler.

The experimental design is given in Section 4.2 and results of the experiments are provided in Section 4.3.

4.2 Experimental Design

Two sets of experiments were conducted on the same networks used by Miller-Hooks and Stock Patterson (2004) for their first set of experiments. Networks of two sizes: 25 nodes and 100 nodes were considered. The average in-degree and out-degree for each node in all the networks is approximately 4, with a minimum of 2 and a maximum of 9. The period of interest is discretized into a number of time intervals. Three numbers of intervals i.e., 60, 120 and 240 were considered for each network. In the first set none of the attributes of the networks were changed from those in Miller-Hooks and Stock

Patterson (2004). This resulted in optimal solutions with the heuristic 100% of the times. Therefore, the second set was devised such that more paths had to be sought at each supply instance, thereby increasing the chances of obtaining a sub-optimal solution. This was achieved by halving the capacities on 50% of the arcs chosen at random. If the resultant capacities were not integral, they were rounded to the next highest integer.

In both sets of experiments, the algorithms were tested and compared for two states of level of supply at the source: light and heavy. For the Quick-TDQFP heuristic and the TDQFP algorithm with no waiting, in a lightly loaded network (i.e. a network with a small level of supply) a quarter of the time intervals in the period of interest were assigned supply. For a heavily loaded network, the supply occurred at half of the time intervals in the time period of interest. A supply of 5 units was assigned to each interval that had supply. Like Miller-Hooks and Stock Patterson (2004), it was assumed that all supply occurs before three quarters of the non-stationary period of interest.

For the TDQFP algorithm with source-only waiting, supply equivalent to the total supply in the no-waiting case was assigned to the first time interval. That is, for the case of 60 time intervals in the period of interest, a supply of 75 units was assigned to the first time interval for the lightly loaded network. Similarly, a supply of 150 units was assigned to the first time interval for a heavily loaded network with 60 time intervals. The same procedure was used with other sets of time intervals.

For each network in the two sets, 10 OD pairs were selected at random. For the second set of experiments, an OD pair was considered only if the problem was feasible with the new arc capacities. That is, if it was found that the network did not have enough capacity to ship all the supply at all time instances, the OD pair was not considered in the experiment.

The code was run for each OD pair. For the Quick-TDQFP heuristic, the total cost (total supply time) at the end of each interval that had a positive amount of supply was compared with that of the TDQFP algorithm to check for a sub-optimal solution. Also, the total run times, number of paths found to ship all the supply and the total time taken to ship all units were recorded. The computation time required to finish 10, 20, 35, 50, 100 and 150 iterations was recorded for each procedure to compare their performance for the same number of iterations. The time taken by the path-finding algorithm was also recorded for each run. Results from these experiments are reported in the next section.

4.3 Experimental Results

For the Quick-TDQFP heuristic, the values of the total cost at the end of each supply interval were found to be equal to those from the exact implementation for the first set of experiments. Thus, the Quick-TDQFP heuristic provided optimal solutions over the entire spectrum of network configurations in the first set of experiments. This may be attributed to the network topology. Even though the situation where the heuristic fails to provide optimal solutions might have occurred, it must have been offset by another node that entered the SE list and caused the sub-optimal label to be updated to optimality before the heuristic terminated. Though, in many cases, the two techniques used different paths for the same supply instance, the paths always arrived at the destination at the same time, thereby resulting in the same cost at the end of each supply instance.

For the second set of experiments, the heuristic provided optimal solutions nearly 90% of the times. When the solutions were sub-optimal, the maximum difference between the optimal and the heuristic solution was less than 1%.

Results from the first set of experiments are provided in Table 4.1.The size of the network is given in the first two columns. The third column indicates the level of loading i.e., the amount of supply that was to be shipped. For each approach, the average run time is reported in the first column. The second column gives the average amount of time taken by the path-finding algorithm. The third column indicates the average number of paths required to ship the supply, and the fourth column gives the average total time taken to ship all units of supply. All reported run times are given in c.p.u. seconds and do not include I/O time, as is common practice in reporting such average run times. All the reported values are average values over runs for 10 OD pairs.

Results in table 4.1 show that the Quick-TDQFP heuristic is much faster than the TDQFP algorithm (when used with TDLTP algorithm for path-finding). The average number of paths employed in the solution found by the Quick-TDQFP heuristic is always slightly higher than that employed by solutions found by the TDQFP algorithm. It is possible that this is due to the existence of multiple optimal paths at each iteration. Since the Quick-TDQFP heuristic updates a previously optimal path at each iteration, there is a high probability that some components of the new path will not change from the prior iteration. Therefore, some portion of the capacity on arcs in the newly determined path would have already been consumed when supply is shipped on it. This would create the need to look for more paths at a particular departure time as compared to the case when paths are solved by beginning from a scratch.

Nodes	Times	Load	TDQFP with TDLTP				Quick-TDQFP				TDQFP with M-Chrono-SPT			
			Total Run Time	Path Finding Time	Path Count	Total Cost	Total Run Time	Path Finding Time	Path Count	Total Cost	Total Run Time	Path Finding Time	Path Count	Total Cost
25	60	LIGHT	1.23	1.22	29.30	986.80	0.40	0.30	30.60	986.80	0.17	0.17	17.78	789.00
25	60	HEAVY	2.96	2.94	64.90	2083.56	0.86	0.67	66.89	2083.56	0.37	0.36	39.10	1968.40
25	120	LIGHT	6.74	6.68	57.30	1612.80	1.49	1.07	57.50	1612.80	0.90	0.89	39.50	1828.40
25	120	HEAVY	13.59	13.52	118.50	3401.60	3.19	4.42	121.80	3401.60	1.73	1.69	79.60	4422.60
25	240	LIGHT	30.87	30.79	115.30	3437.40	6.00	4.39	116.80	3437.40	1.79	1.75	80.30	4765.20
25	240	HEAVY	62.03	61.87	226.88	6571.25	12.42	9.30	231.75	6571.25	3.72	3.66	167.50	10893.13
100	60	LIGHT	12.56	12.54	31.30	1506.90	2.39	1.74	32.60	1506.90	1.68	1.65	22.60	1382.30
100	60	HEAVY	28.06	28.00	64.40	3066.40	5.69	4.44	69.20	3066.40	3.51	3.46	48.30	3025.30
100	120	LIGHT	46.02	45.93	61.27	2880.64	8.19	6.21	62.82	2880.64	6.11	6.03	42.91	3032.00
100	120	HEAVY	95.07	94.86	123.22	5414.22	14.61	10.63	127.89	5414.22	14.03	13.87	100.00	6552.11
100	240	LIGHT	217.23	216.87	126.17	5370.67	27.24	19.04	125.60	5370.67	33.42	33.07	104.17	6714.83
100	240	HEAVY	327.34	324.56	222	7916	39.023	23.73	235	7916	64.29	63.52	198.50	12046.50

Table 4.1: Results from First Set of Computational Experiments

Table 4.2 gives the results from the second set of experiments in terms of the quality of the solutions obtained from the heuristic. The first two columns specify the network topology for the test networks. The third column indicates the type of load assigned to the network. In the fourth column, the frequency of occurrence of a sub-optimal solution per 10 OD pairs is given. That is, a value of 0.1 indicates that the heuristic terminated with sub-optimal values for 1 out of 10 OD pairs. The fifth column gives the relative difference between the total average cost obtained by the heuristic and that by the TDQFP algorithm with TDLTP algorithm as the path finding function. The relative difference was computed by taking the difference between the two averages over 10 runs and dividing it by the average optimal cost. The last column gives the maximum relative difference between the observations. It can be seen, that in each case, the solutions very near to the optimal solution with less than 1% variation.

Nodes	Times	Load	Frequency of sub-optimal solutions (# of OD pairs with sub-optimal solution/10)	Relative difference in Average Total Cost (Average heuristic cost - Average optimal cost)/Average optimal cost	Maximum Relative Difference
25	60	LIGHT	0	0	0
25	60	HEAVY	0.3	0.0016	0.0053
25	120	LIGHT	0	0	0
25	120	HEAVY	0	0	0
25	240	LIGHT	0	0	0
25	240	HEAVY	0.2	0.00017	0.0018
100	60	LIGHT	0	0	0
100	60	HEAVY	0	0	0
100	120	LIGHT	0	0	0
100	120	HEAVY	0	0	0
100	240	LIGHT	0.2	0.00027	0.00046
100	240	HEAVY	0.5	0.00006	0.0009

Table 4.2: Quality of the Heuristic Solution (Experiment Set 2)

Table 4.3 gives the ratio of average run time required by the TDQFP algorithm to the average run time required by the Quick-TDQFP heuristic for each test along with the ratio of the time taken by the path-finding algorithms in each case. It is interesting to note that the ratio of time taken for path finding is higher than the ratio of total run times of both the algorithms in every case. This suggests that the Quick-TDQFP heuristic has some computational burden outside of the path-finding algorithm. Also, it can be observed that as the size of the problem grows, the percentage improvement in run time by using the TDLTP reoptimization algorithm with path diving heuristic grows.

Nodes	Times	Load	Ratio for Total Run Time	Ratio for Time taken for Path-Finding
25	60	LIGHT	3.089	4.074272
25	60	HEAVY	3.446	4.377036
25	120	LIGHT	4.512	6.226324
25	120	HEAVY	4.259	3.05718
25	240	LIGHT	5.145	7.009098
25	240	HEAVY	4.996	6.651068
100	60	LIGHT	5.245	7.201686
100	60	HEAVY	4.927	6.311483
100	120	LIGHT	5.617	7.397675
100	120	HEAVY	6.507	8.922136
100	240	LIGHT	7.974	11.39063
100	240	HEAVY	8.388	13.6772

Table 4.3: Ratios of total run time and time taken for path-finding

It may also be observed that though the worst-case computational complexity of the TDLTP Reoptimization algorithm with the path diving heuristic is worse than that of the original TDLTP algorithm, the reoptimization version performs much better on average. In fact, for bigger problem sizes, the heuristic reoptimization version of the TDLTP algorithm performed better than even the M-Chrono-SPT algorithm, which had the advantage of allowing waiting at the source. The reason the M-Chrono-SPT algorithm does worse in larger networks, or with more supply, is likely that in these cases the number of backwards arcs with finite travel time is quite high. Every time a backward is used, the M-Chrono-SPT algorithm cycles through all the buckets. Thus, a large number of backwards arcs would degrade the performance of the M-Chrono-SPT algorithm. Another observation that can be made is that there is a significant difference in the number of paths used to ship the same amount of supply when using the Quick-TDQFP heuristic as compared with using the TDQFP algorithm with the M-Chrono-SPT algorithm for path-finding. This is probably due to the fact that waiting is allowed at the source node.

The increase in run-times as a function of the number of iterations is shown in Figures 12 and 13. In each of the figures, the units on the Y-axis are c.p.u. seconds and the on the X-axis are the number of iterations. The run-time values for the number of iterations were collected for both the network sizes: 25 node and 100 node with 240 time intervals. The average computation time used for path-finding over 10 runs for 10, 20, 35, 50, 100 and 150 iterations are shown. The significance of improvement obtained from the TDLTP reoptimization heuristic grows with the number of iterations, as indicated by the graphs. It can be seen that as the number of iterations grow in the bigger network, the difference between run-times for the TDLTP Reoptimization algorithm with path diving heuristic and M-Chrono-SPT grows significantly.

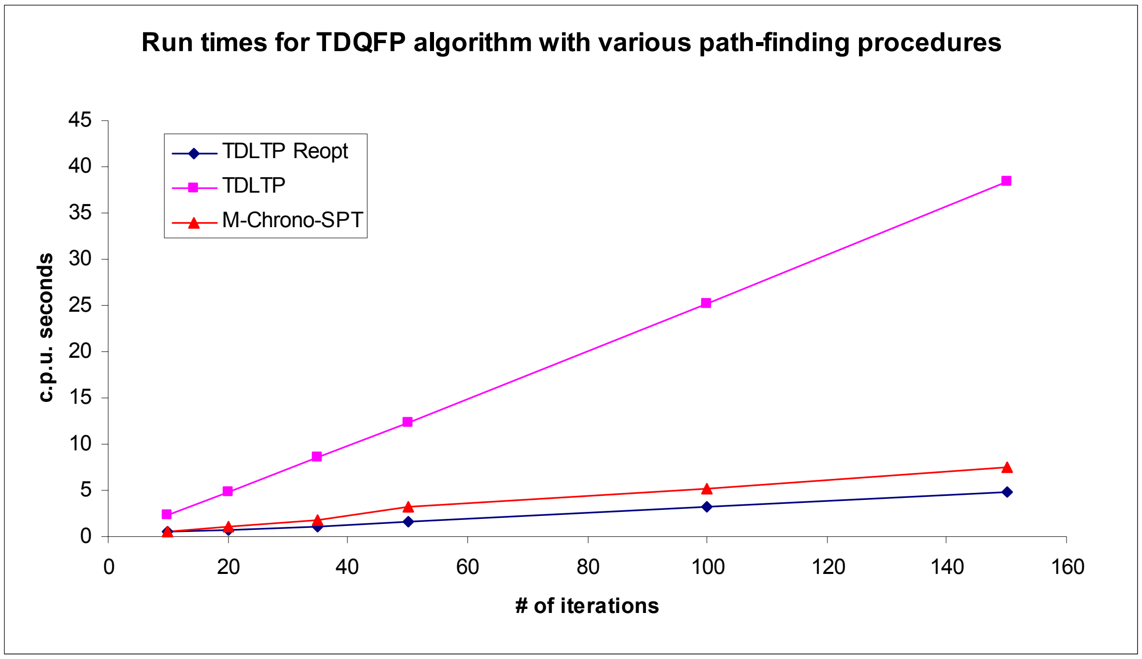


Figure 12: Run Times Vs Number of Iterations (25 nodes, 240 time intervals)

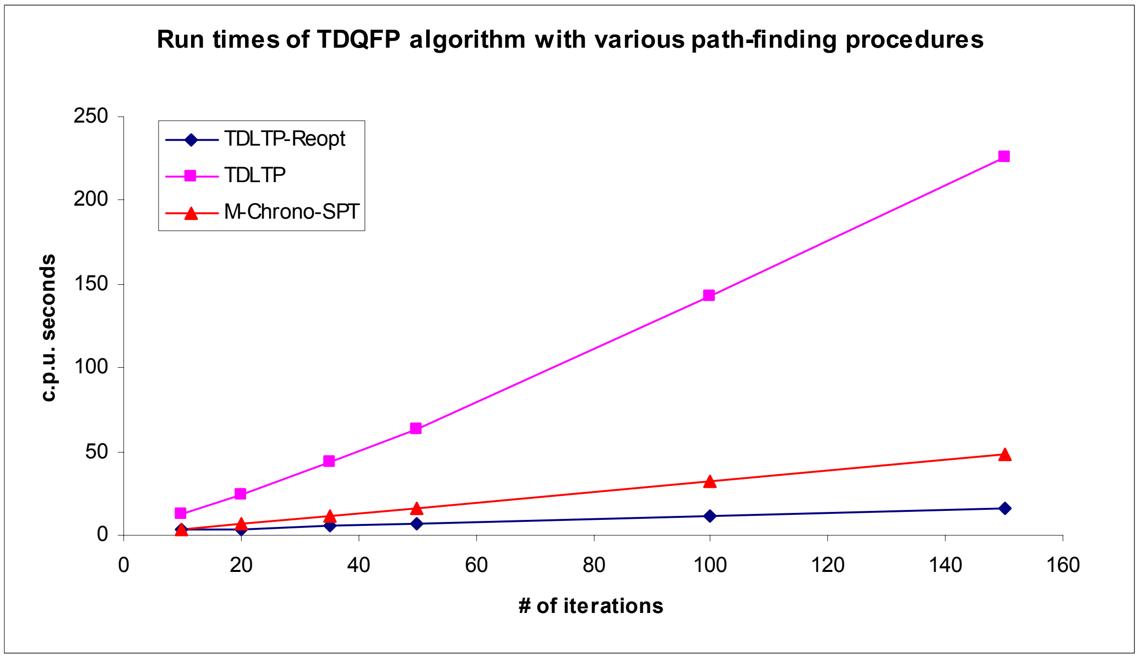


Figure 13: Run Times Vs Number of Iterations (100 nodes, 240 time intervals)

Chapter 5: Further Thoughts and Conclusions

In this thesis, the integral time-dependent quickest flow problem was considered with no-waiting allowed at any of the nodes. A technique intended to improve the computational performance of an existing algorithm based on the successive shortest path approach was proposed. By employing a reoptimization mechanism at each iteration of the algorithm for determining new augmenting paths, the proposed technique can provide significant savings in computation time as observed in numerical experiments. However, it was observed that using reoptimization procedures in residual graphs mandated that the reoptimization algorithm provide optimal results in the presence of zero-sum cycles. No known reoptimization technique was equipped to deal with zero-sum cycles. Therefore, an existing algorithm was modified to create a heuristic that would work in the presence of zero-sum cycles. In the computational experiments it was found that the heuristic provided optimal results 100% of the times for the particular set of tested networks and that it was possible to speed up the existing algorithm between four and eight times depending on the size of the problem. The significance of the computational savings increased with problem size. The proposed conceptual steps may have applicability in improving the performance of any successive shortest path-based approach. Further, a quick exact solution technique based on a modified implementation of an existing path-finding algorithm was proposed to solve the problem when unlimited waiting is allowed at the source. The computational experiments revealed that for problems of smaller size, the exact technique performed much better than the heuristic.

It was observed that the TDLTP Reoptimization algorithm with the PathDive heuristic function was not guaranteed to provide optimal results if the labels for at least one of the constituent nodes of the zero-sum cycle of which the current node was a part could retain its current label by changing the successor node to which its path pointer pointed previously. This would cause the immediate successor node of the current node in the zero-sum cycle to never enter the SE list. The immediate successor node of the current node in the zero-sum cycle was given the term ‘tainted node’. If such a tainted node could be scanned, once the labels for all the constituent nodes of the zero-sum cycles had been set, the labels at the current node would be considered for updating. Thus, if a separate SE list called a tainted list could be maintained, nodes from which are scanned once the regular SE list is empty, the heuristic might terminate with an optimal solution. A node would enter the tainted list, if during some iteration, the function PathDive was to return a value of 1 for this node. If during the subsequent iterations, a label at such a node is updated, it would be removed from the tainted list and added to the regular SE list. However, if some node in the tainted list never enters the SE list, it would still be scanned once the regular SE list is empty, ensuring that all successor nodes of a node are considered in updating its label.

Given that a reoptimization technique can be devised to produce optimal solutions in the presence of zero-sum cycles, it might be worth considering a reoptimization version of the M-Chrono-SPT algorithm. Miller-Hooks and Yang (2004) proposed a reoptimization version for the reverse implementation of the Chrono-SPT algorithm. The only difference in a forward implementation and the existing reverse implementation of the algorithm (Miller-Hooks and Yang, 2004) would be an additional step required by the forward implementation to deal with the node-time pairs that can no longer be reached by any path from the source after the update in travel times. This additional step will be required because while in the reverse implementation of the algorithm each node-time pair has an associated path leading to the destination, there might be node-time pairs that are not reachable by paths from the source when using the forward implementation. Such a reoptimization technique for M-Chrono-SPT might provide significant improvements in computation time.

Appendix A

Some Notation:

$e(t)$ = remaining supply at the source at time t that needs to be shipped

$\hat{t}$ = minimum time at which there is remaining supply at the source

$r_{ij}(t)$ = residual capacity of arc (i, j) at time t

P_l = Vector representing the earliest arrival time path between the source and sink

α_l = arrival time of the earliest arrival time path P_l at the sink

κ_l = capacity of the earliest arrival time path P_l

The TDQFP Algorithm

Step 0

Initialize the following variables: $x = 0, G(x) = G,$ and $e(t) = b_k(t), \forall t \in \{0,...,T\}$

Step 1

Determine $\hat{t}$, where $\hat{t} = \min_{t\in\{0,...,T\}}(t : e(t) > 0)$.

Call the function $TDEAT(k, \hat{t}, T, G(x)),$ whose output contains $\alpha_l, \kappa_l,$ and P_l.

Determine ε, where $\varepsilon = \min[e(\hat{t}), \kappa_l]$. If $\varepsilon = 0$, stop, the problem is infeasible.

Augment ε units of flow along path P_l, i.e., increase $x_{ij}(t)$ by ε, $\forall((i, j), t) \in \mathrm{P}_l$.

Decrease $e(\hat{t})$ by ε.

If $\sum_{t\in\{0,...,T)} e(t) = 0$, **Stop**

Determine the residual capacities for all $(i, j) \in A$ given the flow x.

$$r_{ji}(t) = \begin{cases} u_{ij}(t) - x_{ij}(t) + x_{ji}(t + \tau_{ij}(t)), \forall\{t \in \{0,...T\} \mid t \geq \alpha_i, +\varepsilon, t + \tau_{ij}(t) \leq T \\ 0, \qquad \forall\{t \in \{0,...T\} \mid t < \alpha_i\} \end{cases}$$

Backward arc update:

Add the backward arc (*j, i*) to the residual graph, *G(x)*, if it does not already exist, for each $(i, j) \in A$, such that for some $\{t \in \{0,...T\}$, $x_{ij}(t) > 0$. For all backward arcs, (*j, i*), update the following travel times and residual capacities.

$$\tau_{ji}(t') = \begin{cases} -\tau_{ij}(t), t' = t + \tau_{ij}(t), \forall \{t \in \{0,...T \mid x_{ij}(t) > 0\}, \\ T, \qquad\qquad\qquad\qquad \forall \{t \in \{0,...T\} \mid x_{ij}(t) = 0\} \end{cases}$$

$$r_{ji}(t') = \begin{cases} x_{ij}(t) - x_{ji}(t'), t' = t + \tau_{ij}(t), \forall \{t \in \{0,...T \mid x_{ij}(t) > 0\}, \\ 0, \qquad\qquad\qquad\qquad \forall \{t \in \{0,...T\} \mid x_{ij}(t) = 0\} \end{cases}$$

Repeat **Step 1**

The Quick-TDQFP Heuristic

Step 0

Initialize the following variables: $x = 0, G(x) = G, Q = \varnothing$ and $e(t) = b_k(t), \forall t \in \{0,...,T\}$

Step 1a

Determine $\hat{t}$, where $\hat{t} = \min_{t\in\{0,...,T\}}(t : e(t) > 0)$.

Call function REOPT $(\hat{t}, \Lambda^p, \pi^p, G(x), Q)$, whose output contains $\Lambda, \pi, \alpha_l, \kappa_l,$ and P_l.

Determine ε, where $\varepsilon = \min[e(\hat{t}), \kappa_l]$. If $\varepsilon = 0$, stop, the problem is infeasible.

Augment ε units of flow along path P_l, i.e., increase $x_{ij}(t)$ by ε, $\forall((i,j),t) \in \mathrm{P}_l$.

Decrease $e(\hat{t})$ by ε.

If $\sum_{t\in\{0,...,T)} e(t) = 0$, **Stop**

Step 1b

$Q = \varnothing$

Update residual capacities and residual graph:

For all $((i,j),t) \in P_l$.

Set variables $t_i = t$ and $t_j = t + \tau_{ij}(t)$

$r_{ij}(t_i) = u_{ij}(t_i) - \varepsilon$

If arc $(j,i) \notin G(x)$

```
        add arc $(j,i)$ to $G(x)$

        $r_{ji}(t) = \begin{cases} u_{ji}(t)+\varepsilon, t = t_j \\ 0, \forall\{t \in \{0,...T\} \mid t \neq t_j\} \end{cases}$

        $\tau_{ji}(t) = \begin{cases} -\tau_{ij}(t_i), t = t_j \\ \infty, \forall\{t \in \{0,...T\} \mid t \neq t_j\} \end{cases}$

        $Q = Q + \{(j,i)\}$

    Else If arc $(j,i) \in G(x)$ and $u_{ji}(t_j) = 0,$

        $r_{ji}(t_j) = u_{ji}(t_j) + \varepsilon$

        $\tau_{ji}(t_j) = -\tau_{ij}(t_i)$

        $Q = Q + \{(j,i)\}$

    If $r_{ij}(t_i) = 0,$ $Q = Q + \{(i,j)\}$ and set $\tau_{ij}(t_i) = \infty$

Go to Step 1a.
```

Illustrative Example

The time-dependent network is depicted in Figure A.1. It comprises of 5 nodes and seven arcs connecting those nodes. Node 1 is the source node, i.e., $k = 1$ and node 5 is the sink node i.e., $l = 5$. There is a supply of 4 units at the source at time 0 and of another 2 units at time 2 i.e., $b_1(0) = 4$ and $b_1(2) = 2$. The time interval of interest extends until time 10. The time-dependent travel times and capacities are given in Table A.1.

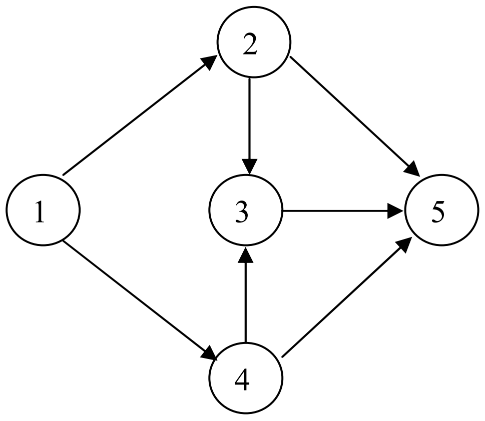


Figure A.1: Example Network

(i,j)	(1,2)	(1,4)	(2,3)	(4,3)	(2,5)	(3,5)	(4,5)
$\tau_{ij}(t)$	$1,\ t=0$ $4,\ t\geq 1$	$2,\ t=0$ $3,\ t\geq 1$	$2,\ t\leq 1$ $3,\ t\geq 2$	$2,\ t\leq 1$ $1,\ t=2$ $3,\ t\geq 3$	$5,\ t\leq 2$ $3,\ t=3$ $5,\ t\geq 4$	$2,\ t\leq 3$ $5,\ t\geq 4$	$2,\ t\geq 1$ $5,\ t\geq 2$
$u_{ij}(t)$	2	$5,\ t=0$ $3,\ t\geq 1$	$5,\ t\leq 1$ $3,\ t\geq 2$	$4,\ t\leq 3$ $2,\ t\geq 4$	5	$2,\ t\leq 4$ $5,\ t\geq 5$	5

Table A.1: Attributes of the Example Network

Following is how the Quick TDQFP algorithm will proceed:

ITERATION 1

Step 0:

$$x=0, e(0)=4, e(2)=2$$
$$Q=\varnothing$$

Step 1a: Path finding and augmenting flow

For the first iteration, it is assumed that the path is made available to the algorithm. This can be done by using any of the path-finding techniques for example, the TDLTP algorithm of Ziliaskopoulos and Mahmasani (1993).

$\hat{t} = 0.$

$\alpha_5 = 5, \kappa_5 = 2,$ and $P_5 = \{((1,2),0),((2,3),1),((3,5),3)\}$

$\varepsilon = \min(\kappa_5, e(0)) = 2$

Augment 2 units of flow along P_5 : $x_{12}(0) = 2, x_{23}(1) = 2, x_{35}(3) = 2$

$e(\hat{t}) = e(\hat{t}) - \varepsilon = 2$

Step 1b: Updating Residual Graph

$Q = \varnothing$

$r_{12}(0) = u_{12}(0) - \varepsilon = 2 - 2 = 0$

Since $(2,1) \notin G(0)$:

add (2,1) to the graph with

$$r_{21}(t) = \begin{cases} 2, t = 1 \\ 0, \forall t \in \{0,...10\} \setminus \{1\} \end{cases}$$

$$\tau_{21}(t) = \begin{cases} -1, t = 1 \\ \infty, \forall t \in \{0,...10\} \setminus \{1\} \end{cases}$$

$Q = Q + \{(2,1)\}$

Similarly for arcs (2,3) and (3,5), $r_{23}(1) = 3$ and $r_{35}(3) = 0$

add arcs (3,2) and (5,3) to the network with the following properties:

$$r_{32}(t) = \begin{cases} 2, t = 3 \\ 0, \forall t \in \{0,...10\} \setminus \{3\} \end{cases}$$

$$\tau_{32}(t) = \begin{cases} -2, t = 3 \\ \infty, \forall t \in \{0,...10\} \setminus \{3\} \end{cases}$$

$$r_{53}(t) = \begin{cases} 2, t = 5 \\ 0, \forall t \in \{0,...10\} \setminus \{5\} \end{cases}$$

$$\tau_{53}(t) = \begin{cases} -2, t = 5 \\ \infty, \forall t \in \{0,...10\} \setminus \{5\} \end{cases}$$

$Q = \{(1,2),(3,5),(2,1),(3,2),(5,3)\}$

$\tau_{12}(0) = \infty, \tau_{35}(3) = \infty$

The updated residual graph with backwards arcs is shown in figure A.2. The curved arcs represent the backwards arcs and the dashed arcs indicate the arcs on which the residual capacity is now zero (travel time is infinity) for some time instance.

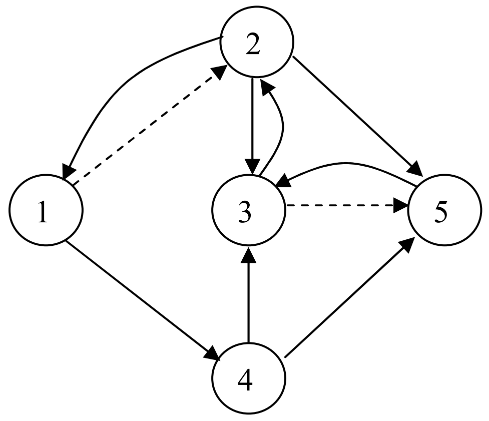


Figure A.2: Residual Graph after Iteration 1

ITERATION 2

Effectively, this is the step where the Quick TDQFP algorithm differs from the TDQFP algorithm. Instead of calling the same path finding function as in the first iteration which considers the residual graph as a new graph, the Quick TDQFP calls the REOPT function which reoptimizes the solution for the previous graph based on the list of changes (sets B and Q) made to the residual graph. The steps for the second iteration follow.

Step 1a:

$\hat{t} = 0.$

Call REOPT$(0, \Lambda^p, \pi^p, G(x), Q)$

which gives us $\alpha_5 = 6, \kappa_5 = 2,$ and $P_5 = \{((1,4),0),((4,3),2),((3,2),3),((2,5),1)\}$

$\varepsilon = \min(\kappa_5, e(0)) = 2$

Augment 2 units of flow along P$_5$: $x_{14}(0) = 2, x_{43}(2) = 2, x_{23}(1) = 0, x_{25}(1) = 2$

$e(\hat{t}) = e(\hat{t}) - \varepsilon, i.e., e(0) = 2 - 2 = 0.$

Step 1b:

$Q = \varnothing$
$r_{14}(0) = 3, r_{43}(2) = 2, r_{32}(3) = 0, r_{25}(1) = 3.$
The residual capacities are calculated and backwards arcs added in the same way as in Iteration 1 except for arc (3,2) for which the steps are as following
$t_3 = 3, t_2 = 1$
Since $(2,3) \in G(x)$:
$r_{23}(1) = r_{23}(1) + \varepsilon, i.e., r_{23}(1) = 3 + 2 = 5$
$\tau_{23}(1) = -\tau_{32}(3), i.e., \tau_{23}(1) = -(-2) = 2.$
$Q = Q + \{(2,3)\}$
The step ends with:
$Q = \{(3,2),(4,1),(3,4),(2,3),(5,2)\}$
$\tau_{32}(3) = \infty$

The updated residual graph after the second iteration can be seen in Figure A.3

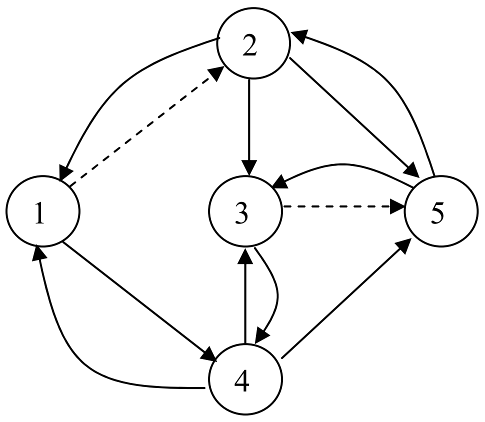


Figure A.3: Residual Graph after Iteration 2

ITERATION 3

Step 1:

$\hat{t} = 2.$

Since Iteration $\neq$ 0, Call REOPT$(2, \Lambda^p, \pi^p, G(x), Q)$

which gives us $\alpha_5 = 10, \kappa_5 = 3,$ and $P_5 = \{((1,4),2),((4,5),5)\}$

$\varepsilon = \min(\kappa_5, e(0)) = 2$

Augment 2 units of flow along P$_5$: $x_{14}(2) = 2, x_{45}(5) = 2$

$e(\hat{t}) = e(\hat{t}) - \varepsilon, i.e., e(2) = 2 - 2 = 0.$

Since $\sum_{t \in \{0,\ldots 10\}} e(t) = 0$, STOP

The algorithm terminates with the following flow solution.

Send

- 2 units along {((1, 4), 0), ((4, 3), 2), ((3, 5), 3)} which arrives at node 5 at time 5
- 2 units along {((1, 2), 0), ((2, 5), 1)} which arrives at node 5 at time 6
- 2 units along {((1, 4), 2), ((4, 5), 5)} which arrives at node 5 at time 10

The time-dependent quickest flow time is 10, and the total time taken to send the supply is 12+10+16 = 38.

References

Ahuja, R. K., T. L. Magnati, J. B. Orlin. 1993. Network Flows: Theory, Algorithms and Apllications. Prentice Hall, Inc., New Jersey.

Anderson, E. J., P. Nash., A. B. Philpott. 1982. A Class of Continuous Network Flow Problems. *Mathematics of Operations Research*. **7** 501-14.

Bellman, R., 1958. On a Routing Problem. *Quarterly Applied Mathematics*. **16** 87-90.

Bertsimas, D., J. N. Tsitsiklis. 1997. *Introduction to Linear Optimization*. Athena Scientific, Belmont, Massachusetts.

Bertsekas, D. P. 1991. *Linear Network Optimization: Algorithms and Codes*. MIT Press, Cambridge, Massachusetts.

Burkard, R., K. Dlaska, B. Klinz. 1993. The Quickest Flow Problem. *Methods and Models of Operations Research*. **37** 31-58.

Cai, X., D. Sha, C. K. Wong. 2001. Time-varying Minimum Cost Flow Problems. *European Journal of Operational Research*. **131** 352-374.

Chabini, I., 1998. Discrete Dynamic Shortest Path Problems in Transportation Applications: Complexity and Algorithms with Optimal Run Time. *Transportation Research Record* **1645** 170-175.

Cooke, K., E. Halsey. 1966. The Shortest Route through a Network with Time-Dependent Intermodal Transit Times. *Journal of Mathematical Analysis and Applications* **14** 493-498.

Dijkstra, E. W., 1959. A Note on Two Problems in Connexion with Graphs. *Numerishe Matematik* **1** 269-271.

Dreyfus, S. E., 1969. An Appraisal of Some Shortest Path Algorithms. *Operations Research* **17**, 395-412.

Ford, L., D. Fulkerson, 1962. *Flows in Networks*. Princeton University Press, New Jersey.

Halpern, J. 1979. A Generalized Dynamic Flows Problem. *Networks* **9** 133-167.

Hoppe, B. 1995. Efficient Dynamic Network Flow Algorithms. Technical Report TR95-1524. Department of Computer Science, Cornell University, Ithaca, New York.

Hoppe, B., E. Tardos. 2000. The Quickest Transshipment Problem. *Mathematics of Operations Research* **25** 36-62.

Klingman, D., J. Mote. 1982. A Multi-period Production, Distribution and Inventory Planning Model. *Advances in Management Studies* **1** 56-76.

Miller-Hooks, E., S. Stock Patterson. 2004. On Solving Quickest Time Problems in Time-Dependent, Dynamic Networks. *Journal of Mathematical Modelling and Algorithms* **3** 39-71.

Miller-Hooks, E., B. Yang. 2004. Updating Paths in Time-Varying Networks given Arc Weight Changes. Forthcoming in *Transportation Science*

Opasanon, S. 2004. On Finding Paths and Flows in Multi-Criteria, Stochastic and Time-Varying Networks. Dissertation Thesis, Department of Civil and Environmental Engineering, Univeristy of Maryland, College Park, Maryland.

Orda, A., R. Rom. 1990. Shortest-Path and Minimum-Delay Algorithms in Networks with Time-Dependent Edge-Length. *Journal of the Association for Computing Machinery* **37** 607-625.

Pallottino, S., M. Scutella. 1998. Shortest Path Algorithms in Transportation Models: Classical and Innovative Aspects. In *Equilibrium and Advanced Transportation Modeling* edited by Marcotte, P. and Nguyen, S., Kluwer Academic Publishers, Boston, 245-281.

Ziliaskopoulos, A., H. Mahmassani. 1993. Time-Dependent, Shortest-Path Algorithm for Real-Time Intelligent Vehicle Highway System Applications. *Transportation Research Record* **1408** 94-100.