DTU Compute
Department of Applied Mathematics and Computer Science

Approximation Algorithms for Replenishment Problems with Fixed Turnover Times

Hans Henrik Hermansen

Approximation Algorithms for Replenishment Problems with Fixed Turnover Times

Master Thesis
June, 2025

By
Hans Henrik Hermansen
Supervisors
Inge Li Gørtz & Philip Bille

Cover photo: Vibeke Hempler, 2012

Abstract

In this thesis, we study a problem called Replenishment Problems with Fixed Turnover Times (RFTT). The problem was first presented in [1] and combines routing and scheduling, two already hard classes of problems. It models a network of clients who each store a commodity. The amount each client has stored depletes over time. A client should never run out of this commodity, so we must ensure that we visit each client and refill their stock often enough. How quickly stock depletes varies from client to client, and it is allowed to visit and refill a client way before the clients stocks runs out. In this thesis we, like in [1], study both the Min-Avg and Min-Max versions of the problem, where the objectives are to minimize average cost of a route and minimize the cost of the longest route respectively. We cover the approximation algorithms on trees for both metrics given by [1] and we improve the complexity of their pseudo-polynomial algorithm for Min-Avg on a path. We then give a polynomial time algorithm for Min-Max on cycles, a 4-approximation for Min-Avg on cycles and insight into how a pseudo-polynomial algorithm for Min-Avg on cycles might be achieved. Finally we combine the ideas from the tree algorithms and cycle algorithms to give approximation algorithms for graphs, where all cycles are disjoint. On these graphs we give an algorithm for finding optimal routes, a $6 + N_C$-approximation for Min-Max, where N_C is the number of cycles in a graph, and a 4-approximation for Min-Avg.

Contents

1 Introduction

In this thesis, we study the problem of Replenishment Problems with Fixed Turnover Times (RFTT) under the optimization metrics of Min-Avg and Min-Max. The problem was first defined in [1] and combines routing and scheduling, two already hard classes of problems. It models a very natural situation. We have a network of clients who each store a commodity, which we supply. The supply of each client depletes over time, but the speed with which their supply empties varies from client to client. The number of days it takes for a client to run out, is that client's turnover time, and this number is fixed. We must ensure that no client ever starts a day with no supply, which we do by routing vehicles each day to refill some subset of the clients. Refilling a client before the client's stock runs out is allowed and it is easy to imagine instances, where doing so is optimal. This feature contributes greatly to the difficulty of working with RFTT. For Min-Avg the objective is to minimize the average cost of a route and for Min-Max the objective is to minimize the cost of the longest route taken.
Apart from defining the problem, they in [1] also give constant factor approximations for trees and $O(\log n)$-approximations for general graphs for both metrics. Additionally they provide an exploration of the hardness of RFTT and a pseudo-polynomial for Min-Avg on paths. There also exists a $O(\log \log n)$-approximation for Min-Avg on general graphs given by [2].
We do not cover the approximation algorithms for general graphs, but instead focus on adding to the types of graphs for which we have constant factor approximations. We therefore cover the 2-approximation for Min-Avg on trees and the 6-approximation for Min-Max on trees given by [1]. We also improve upon the complexity of their pseudo-polynomial algorithm for Min-Avg on paths. We then study RFTT on cycles. Here we give a $O(n)$ algorithm for solving Min-Max and a 4-approximation for Min-Avg. Additionally we give insights into how a pseudo-polynomial for Min-Avg on cycles could be achieved. Next we define a family of graphs, we call cycle-trees, which are graphs where all cycles are disjoint. For Min-Max we give a $6 + 2N_C$-approximation for cycle-trees, where N_C denotes the number of cycles in a graph. For Min-Avg we combine the 2-approximation for trees with our 4-approximation on cycles to give a 4-approximation for cycle-trees. To facilitate these we also give an algorithm for finding the optimal route on cycle-trees.
Finally we use our results to give multiple paths towards both finding new results and improving upon the ones we have presented.

2 Preliminaries

In this thesis, we will assume basic knowledge of both graph theory and algorithms. In this section we define some basic graph theory notation and concepts, that we will need throughout the entire thesis. Furthermore we will also provide a definition of an approximation algorithm.

2.1 Graph Theory

First we define a weighted graph

Definition 2.1. *A weighted graph G of size n is a tuple (V, E, c). Here $V = [n]$ is the set of vertices, $E \subseteq \{\{x, y\} \mid x, y \in V \wedge x \neq y\}$, a set of unordered pairs, is the set of edges and $c : E \mapsto \mathbb{R}_+$ is the cost function.*

For our purposes we will only deal with simple and undirected graphs. The next important concept to introduce is a walk.

Definition 2.2. *Given a graph $G = (V, E, c)$, a walk w of length l is a sequence of vertices and edges $w = (v_1, e_1, v_2, e_2, \ldots e_l, v_{l+1})$. The cost of a walk is denoted $c(w)$ and is defined as $c(w) = \sum_{e \in w} c(e)$.*

This leads to a more specific type of walk that is also very relevant for our problem.

Definition 2.3. *Given a graph G, a route r is a walk that begins and ends in the same vertex. Meaning that $v_1 = v_{l+1}$*

We will also need to describe the distance between to vertices. For vertices v, w we let the distance $d(v, w)$ be the length of the shortest path from v to w

2.2 Approximation Algorithms

In this thesis we will mainly be dealing with approximation algorithms. We will therefore define what an approximation algorithms is. We use approximation algorithms to approximate the solutions to very hard, usually NP-hard, optimization problems. We approximate the solutions to these problems, as finding an optimal solutions takes exponential time. So in order for an approximation algorithm to be of any use, it should run in polynomial time. Furthermore we want to be sure how much worse than the optimal solution the approximation will be. This will let us compare the quality of different approximation algorithms. There exists approximation algorithms for both maximization and minimization problem, however since the focus of this paper is a minimization problem that is what we will focus on. This motivates the following definition

Definition 2.4. *Let I be a problem instance and A an algorithm. Then let OPT be the cost of the optimal solution to I and $A(I)$ the cost of the solution obtained by using A on I. Then A is a α-approximation if*

- *The solution produced by A is feasible*
- *A runs in polynomial time.*
- $A(I) \leq \alpha \cdot OPT$

3 Problem Definition

We will now formally introduce the problem that is the focus of this thesis.

Definition 3.1. *An instance I^R of Replenishment Problem with Fixed Turnover Times (RFTT) is a tuple (G, u) where G is a weighted graph and u is a designated root vertex. Furthermore each vertex v has an associated turnover time τ_v.*

Note that the notation for this definition is slightly different to the one given by [1], however they are functionally the same.
Formally, a solution to an instance I^R is an infinite sequence of routes $(r_1, r_2, r_3, \dots)$ where route r_t specifies the route taken on day t. This representation however is not practical so we will only look at solutions that repeat after a point. As a result we define a solution as follows

Definition 3.2. *Given an instance I^R of RFTT, a solution S is a finite sequence $(r_1, r_2, r_3, \dots, r_L)$ of routes. The length of S we denote L. The route taken on day t is r_i where $i \equiv 0 \mod L$.*

This representation is still in many cases impractical, as it will often take exponential space and thus exponential time to describe, even though it might be possible to calculate a schedule for the vertices in polynomial time. To accommodate this we will allow for the following representation of a solution.

Definition 3.3. *A solution to an instance of RFTT may be represented as two functions $h(v) : V \mapsto \mathbb{N}$ and $p(v) : V \mapsto \mathbb{N}$. Here $h(v)$ is the first day v is visited and $p(v)$ is the number of days between each consecutive visit.*

This is the representation used by all the approximation algorithms presented in this thesis. We will assume that the route taken on a given day is calculated on the day. In order for this to be practical it cannot take too long to find this route. We must therefore provide a way to find a route within a reasonable time, when representing a solution in this manner. This method of finding a route can differ from solution to solution, as it can depend on the input graph. This motivates the following definition

Definition 3.4. *Let S be a solution of an instance of RFTT represented by functions $h(v), p(v)$. Now let t be a day and V_t be the vertices that S must visit on day t. Then S is polynomial if is provides a method of calculating a route, that visits all of V_t, in $O(P(n))$ time, where $P(n)$ is some polynomial.*

Whenever an algorithm finds a solution in polynomial time, we must ensure that the solution it produces is polynomial.

A solution is only feasible if all vertices are visited enough. To capture this we first define what it means for a vertex to be visited enough.

Definition 3.5. *Let I^R be an instance of RFTT and S a solution to I^R. A vertex v is feasible in S if for any sequence of consecutive τ_v days, v is visited at least once.*

For the compact representation of a solution this means that a vertex is feasible if $h(v) \leq \tau_v$ and $p(v) \leq \tau_v$. We can now define a feasible solution.

Definition 3.6. *Let I^R be an instance of RFTT and let S a solution to I^R. Then S is feasible if all vertices are feasible in S.*

Now that we have defined problem instances and feasible solutions we can look at the two different optimization metrics for RFTT we will cover in this thesis. These are the Min-Avg and Min-Max metrics. For Min-Avg the goal is to minimize the average cost per day and for Min-Max it is to minimize the cost of the most expensive day. For Min-Avg we get the following definition.

Definition 3.7. *Let I^R be an instance of RFTT and S a solution to I^R. Now let L be the length of S. Then the Min-Avg objective value of S, denoted as $c(S)$, is $\frac{1}{L}\sum_{i=1}^{L} c(r_i)$.*

and correspondingly for Min-Max.

Definition 3.8. *Let I^R be an instance of RFTT and S a solution to I^R. Now let L be the length of S. Then the Min-Max objective value of S, denoted as $c(S)$, is $\max_{i\in[L]} c(r_i)$.*

4 Hardness

In this chapter, we briefly cover one of the hardness results given by [1]. As they mention, the time horizon for this problem is infinite, so it is very tricky to properly classify its hardness. However we can still show that the problem is NP-hard. One easy simplification of RFTT is to make every turnover time equal to 1. In this case the solution to Min-Avg and Min-Max is the same, as every vertex must be visited every day. Here the problem boils down to simply finding the cheapest route which visits every vertex. We will show that this then solves the Metric Traveling Salesman Problem (m-TSP), which is a known NP-Hard problem. We first give a short definition of m-TSP.

Definition 4.1. *An instance of m-TSP, I^{TSP}, is a tuple $(G.u)$ where G is a complete weighted graph and u is a start vertex. Furthermore c satisfies the triangle inequality. This means that for any three vertices v, y, w in G it holds that $c(\{v, y\}) \leq c(\{v, w\}) + c(\{w, y\})$. A solution to I^S is a route, r, which starts in u and visits every vertex. The objective is to minimize $c(r)$.*

With this we will now present and prove the following Theorem.

Theorem 4.1. *There exists a polynomial time reduction from m-TSP to RFTT. Thus RFTT is at least as hard as m-TSP and therefore NP-Hard.*

Proof. Let $I^{TSP} = (G, u)$ be an instance of m-TSP. Now construct an instance of Min-Avg RFTT, $I^R = (G, u)$, where for all $v \in V$ we set $\tau_v = 1$. Let S be an optimal solution to I^R and r_1 the route taken by S on day 1. We know that r_1 starts in u and visits every vertex. Since S is optimal, we also know that it is not possible to find a cheaper route that does the same. Now construct a new route in the following way. Follow r_1 until the next vertex is one that has already been visited. Now follow the edge to the next vertex in r_1 that has not yet been visited. Repeat this process until all vertices have been visited. Now follow the edge to u. Now r only visits each vertex once. When constructing r we may have replaced a path between vertices v, w with the edge between them. Since c satisfies the triangle equality, this has not increased the cost of the route. As we know r_1 is optimal we can conclude that $c(r_1) = c(r)$. So r is a route that visits every vertex in G once with minimum cost. This is exactly the solution to I^{TSP}. □

As the hardness of RFTT is not the focus of this thesis, we will not discuss this further. We suggest reading [1] as they go to much greater lengths exploring the hardness of RFTT. They compare the hardness of RFTT to many other NP-Hard problems and even prove that Min-Max is NP-Hard on star graphs.

5 Approximation Algorithms on Trees

We will now cover the approximation algorithms on trees given by [1] for both Min-Avg and Min-Max. Furthermore we give a new pseudo-polynomial algorithm for Min-Avg on path, that improves the time complexity of the one given by [1]. Every algorithm, Theorem, Lemma and proof in the first two sections of this chapter closely follow the ones presented in [1]. Firstly we will introduce an observation about feasible solutions in general.

Lemma 5.1. *Let I^R and I'^R be instances of RFTT where $G = G'$. Now for all $v \in V$ let $\tau_v \leq \tau_{v'}$. Then every solution feasible solution to I^R is also a feasible solution to I'^R.*

Proof. Let S be a feasible solution to I^R and a solution to I'^R. For a vertex v' to be feasible in S it must be visited at least once in any interval of $\tau_{v'}$ days. Since $\tau_{v'} \geq \tau_v$, any interval of $\tau_{v'}$ days must contain an interval of τ_v day. As v is feasible in S it must have been visited in that interval and therefore so will v'. Therefore all $v' \in V'$ are feasible in S and thus S is also a feasible solution to I'^R □

We can now introduce an important Lemma used throughout this entire thesis.

Lemma 5.2. *Let I^R be an instance of RFTT. Now let $I^{\frac{R}{2}}$ be a copy of I^R where all turnover times have been rounded down to nearest power of two. Then $OPT^{\frac{R}{2}} \leq 2OPT^R$ for both Min-Avg and Min-Max.*

Proof. Let I^{2R} be a copy of I^R where all turnover times have been rounded up to closest power of two and let S^R be a feasible solution to I^R. From Lemma 5.1 we know that S^R must also be a feasible solution to I^{2R}. This means that an optimal solution to I^R is also a solution to I^{2R} and we then get $OPT(I^R) \geq OPT(I^{2R})$ for both metrics.
We can also turn any feasible solution S^{2R} to I^{2R} into a feasible solution $S^{\frac{R}{2}}$ to $I^{\frac{R}{2}}$ in the following way. On day t in $S^{\frac{R}{2}}$ visit everything that is visited by S^{2R} on day $2t$ and $2t+1$. Since the cost of any day in $S^{\frac{R}{2}}$ is at most the sum of the cost of two days in S^{2R} it is clear that $OPT(I^{\frac{2}{R}}) \leq 2OPT(I^{2R}) \leq 2OPT^R$ on the Min-Max metric. Now for Min-Avg we get that the cost of $S^{\frac{R}{2}}$ can be bounded in the following way.

$$c(S_{\frac{R}{2}}) = \frac{1}{\frac{L}{2}} \sum_{i=1}^{\frac{L}{2}} c(r_i^{\frac{R}{2}}) \leq \frac{2}{L} \sum_{i=1}^{\frac{L}{2}} c(r_{2i}^{2R}) + c(r_{2i+1}^{2R}) = 2\left(\frac{1}{L}\sum_{i=1}^{L} c(r^{2R}i)\right) = 2OPT^{2R}$$

With this we get $OPT^{\frac{R}{2}} \leq c(S^{\frac{R}{2}}) \leq 2OPT^{2R} \leq 2OPT^R$ for the Min-Avg metric as well. □

When working with trees we know that every path is unique. This means that a vertex v is also visited any time we visit any of its descendants. Therefore we in any feasible solution visit v at least as often as we visit any of its descendants. As a result we can wlog. assume that τ_v is no larger than the minimum turnover time among its descendants. This also means that along any root-to-vertex path, the turnover times will be in non-decreasing order. This insight helps us bound how often an edge must be used. In the following let $D(e)$ describe the descendant vertices of the edge e.

Definition 5.1. *Let I^R be an instance of RFTT. For every edge $e \in E$ we define*

$$q(e) = \min_{v \in D(e)} \tau_v$$

Now with the assumption we make on trees, $q(e)$ is simply the turnover time of the vertex incident to e that is furthest from the root. We now use this to lower bound the average cost per day. This is an important lower bound so we will denote it $\Gamma(I^R)$. This leads us to present the following Lemma.

Lemma 5.3. *Let I^R be an instance of RFTT where G is a tree. Then the average cost of a day is at least*

$$\Gamma(I^R) = 2 \sum_{e \in E} \frac{c(e)}{q(e)}$$

Proof. Let S be a feasible solution to I^R with length L. Any edge e must be traversed at least $2L$ times during first $q(e) \cdot L$ days. For this to be true S must traverse e at least $2 \cdot \left\lceil \frac{L}{q(e)} \right\rceil$ times. Since this is true for every edge we can lower bound the average cost as follows

$$\frac{2}{L} \sum_{e \in E} c(e) \left\lceil \frac{L}{q(e)} \right\rceil$$

We can now use this to lower bound the average cost per day of any solution independent of L.

$$\frac{2}{L} \sum_{e \in E} c(e) \left\lceil \frac{L}{q(e)} \right\rceil \geq \frac{2}{L} \sum_{e \in E} c(e) \frac{L}{q(e)} \geq 2 \sum_{e \in E} \frac{c(e)}{q(e)} = \Gamma(I^R)$$

□

5.1 Min-Avg 2-approximation

We are now ready to present the 2-approximation for Min-Avg given by [1].

Algorithm 1

```
  Round down every turnover time to nearest power of two
  For all $v$ let $h(v) = p(v) = \tau_v$
  Return $h(v)$ and $p(v)$
```

Which leads to present the following Theorem.

Theorem 5.1. *Algorithm 1 is a 2-approximation for Min-Avg on trees.*

Proof. **Feasibility** Let I^R be an instance of RFTT where G is a tree and $I^{\frac{R}{2}}$ be the instance where all turnover times are rounded down to nearest power of two. Now let S be the solution found by algorithm 1 to instance $I^{\frac{R}{2}}$. Since $h(v) = p(v) = \tau_v$ for all $v \in V$, we know that S is feasible. We from Lemma 5.1 then get that S is also a feasible solution for I^R.

Bound Since $h(v) = p(v) = \tau_v$ for all $v \in V$, we know that v is visited on any day t where $t \equiv 0 \mod \tau_v$. As all turnover times are powers of two ,this means that on day τ_{max} all vertices are visited so $L = \tau_{max}$. For an edge $e \in E$, we know that all of its descendants have turnover times larger than or equal to $q(e)$. This means that e is only traversed on days t where $t \equiv 0 \mod q(e)$. As a result we can calculate $c(S)$ as follows

$$c(S) = \frac{1}{\tau_{max}} \sum_{i=1}^{\tau_{max}} c(r_i) = \frac{1}{\tau_{max}} \sum_{e \in E} 2 \cdot c(e) \frac{\tau_{max}}{q(e)} = 2 \sum_{e \in E} \frac{c(e)}{q(e)} = \Gamma(I^{\frac{R}{2}})$$

Since $\Gamma(I^{\frac{R}{2}})$ is a lower bound we then from this and Lemma 5.2 get that $c(S) = \Gamma(I^{\frac{R}{2}}) = OPT^{\frac{R}{2}} \leq 2OPT^R$.

Complexity Rounding down all turnover times and then assigning $h(v), p(v)$ values is clearly $O(n)$. Finding the route on a given day is easy as it is simply a traversal of a subtree so this is also $O(n)$. This algorithm therefore has complexity $O(n)$ and produces a polynomial solution. □

5.2 Min-Max 6-approximation

We will now cover the 6-approximation for Min-Max given by [1]. As this algorithm is quite a bit more complex than the one for Min-Avg we will start by presenting the main idea of the algorithm. The main idea is to spread out visiting all vertices with turnover time i among the i days of an interval. So instead of visiting all vertices with turnover time 4 on days that are congruent 0 mod 4, we spread them out among days that are congruent 0, 1, 2 and 3. This we will do by recursively splitting up a traversal of G into smaller and smaller walks while ensuring that the resulting walks are of roughly equal cost. To do this we need to define a well-balanced split.

Definition 5.2. *Let I^R be an instance of RFTT where G is a tree. Noe let w be a walk on G. Then a well-balanced split of w consists of two walks $w_{pre} = (v_1, e_1, v_2, \ldots, v_k)$ and $w_{post} = (v_{k+1}, e_{k+1}, v_{k+2}, \ldots, e_l, v_{l+1})$ where the following holds*

$$\sum_{e \in w_{pre}} \frac{c(e)}{q(e)} \leq \frac{1}{2} \sum_{e \in w} \frac{c(e)}{q(e)} \text{ and } \sum_{e \in w_{post}} \frac{c(e)}{q(e)} \leq \frac{1}{2} \sum_{e \in w} \frac{c(e)}{q(e)}$$

The algorithm for Min-Max on trees consists of two parts. A main algorithm and a recursive algorithm. The main algorithm makes basic calculations and provides the input to the recursive algorithm. The recursive then calculates the $h(v), p(v)$ values for each vertex based on the input from the main algorithm. The main algorithm is as follows

Algorithm 2

```
Round down all turnover times to nearest power of two.
Let $W$ be a DFS traversal of $G$.
Calculate $q(e)$ for all $e \in E$
Use $MaxTreeSchedule(W, 1, 0)$ to calculate $h(v)$ and $p(v)$
return $h(v)$ and $p(v)$
```

Next we give the description of the recursive algorithm which we, like in [1], name *MaxTreeSchedule* for ease of reference.

Algorithm 3 $MaxTreeSchedule(W, t, k)$

```
if $W$ only contains a single vertex $v$ then
    if $\tau_v \geq 2^k$ then
        Set $h(v) = t$ and $p(v) = \tau_v$
        Return $h(v)$ and $p(v)$
    end if
end if
Let $X = \{v \mid v \in W \wedge \tau_v = 2^k\}$ and $Y = \{e \mid e \in W \wedge q(e) = 2^k\}$
For all $v \in X$ set $h(v) = t$ and $p(v) = \tau_v$
Contract all edges in $Y$ in $W$. So $W' = W/Y$
Create $W_0$ and $W_1$ as a well-balanced split of $W'$
Compute $MaxTreeSchedule(W_0, t, k + 1)$ and $MaxTreeSchedule(W_1, t + 2^k, k + 1)$
```

Note that due to how walks are split, a vertex may have its $h(v), p(v)$ values assigned by multiple calls. When this happens it is simply the last value assigned which is the final value. This does not affect the analysis in any way. To help illustrate how the walks are split and $h(v), p(v)$ values are assigned we provide an example run of $MaxTreeSchedule$ in figure 5.1.

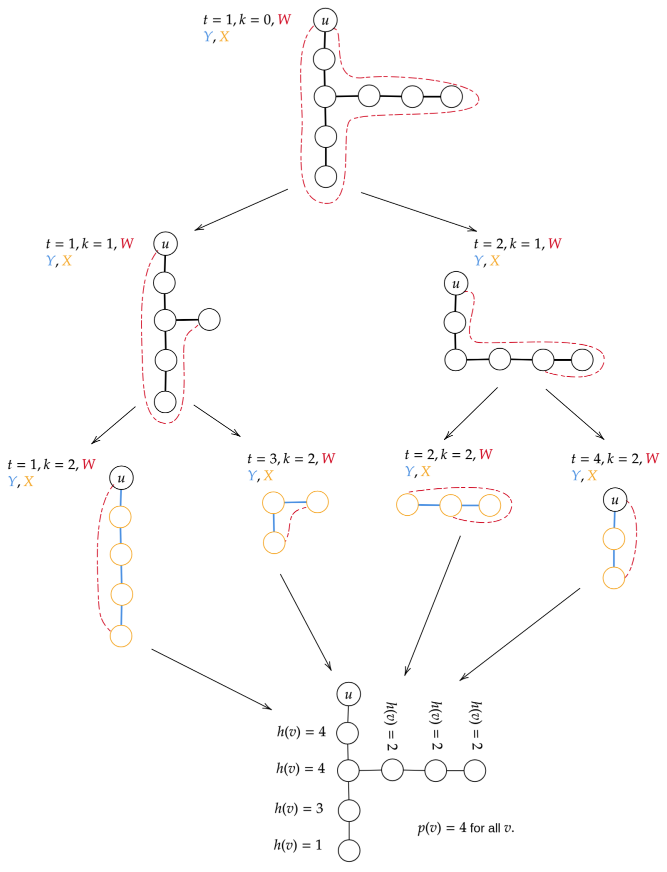

Figure 5.1: A run of $MaxTreeSchedule$ on a simple graph, where all turnover times are equal to 4. Every edge weight is equal to 1.

The analysis of algorithm 2 essentially boils down to the analysis of $MaxTreeSchedule$. To analyze $MaxTreeSchedule$ we note the following. On any day where a vertex v is visited, we will also visit all vertices assigned by an ancestor call to the call that assigned v. We denote a call, which makes no calls to $MaxTreeSchedule$ itself, as a leaf call. So on a day with maximum cost we must visit a vertex assigned by a leaf call. Therefore we will look to bound the cost of visiting vertices assigned by a leaf call and its ancestor calls. To do this we will prove properties that hold in any call and apply this to a leaf call. As we will be proving properties of calls to $MaxTreeSchedule$, we will refer to X and Y as the sets in the description of $MaxTreeSchedule$.

Lemma 5.4. *In any call to $MaxTreeSchedule$, Y spans X in W.*

Proof. First we note that W does not contain any vertices with turnover time less than 2^k. Any such vertex would be incident to an edge e with $q(e) < 2^k$ and would have been contracted. Due to our assumption about the turnover times on G we know that on any path P between vertices v, w it holds that $q(e) \leq \max(\tau_v, \tau_w)$ for all $e \in P$. This means that any path between two vertices in X can only use edges with $q(e) = 2^k$ which are all in Y. Thus the edges Y connects all vertices in X. $\square$

We will now let Y^{anc} be the set of edges contracted by ancestor calls.

Lemma 5.5. *In any call to $MaxTreeSchedule$, $Y^{anc} \cup Y$ spans X in G.*

Proof. We know that Y spans X in W where Y^{anc} has been contracted. Now select an edge in Y^{anc} and uncontract it. If this splits a vertex not in X, X is still connected. If the vertex is in X this splits X into two components connected by the uncontracted edge. So X remains connected using edges in $Y^{anc} \cup Y$. Repeat this process until all edges in Y^{anc} are uncontracted and X is still connected. $Y^{anc} \cup Y$ now clearly corresponds to a subgraph of G that connects X. $\square$

Now let X^{anc} be the set of vertices assigned by ancestor calls. If we apply Lemma 5.5 to all ancestor calls we see that $Y^{anc} \cup Y$ contains $X^{anc} \cup X$ as connected components. We cannot guarantee that these components are connected and as seen the example in figure 5.1 Y is not even necessarily connected to the root. We will fix both of these problems using only a singe path.

Lemma 5.6. *In any call to $MaxTreeSchedule$, if Y is non empty, then the vertex $v \in X$ closest to the root in G is an ancestor to all vertices in W with turnover time larger than 2^k.*

Proof. From our assumption on trees we know that all descendants of a vertex w must have turnover times larger than or equal to τ_w. This, combined with the fact that W is traversing in a depth first manner, means that any vertex with turnover time larger than 2^k must be a descendant of a vertex in X. Since v is closest to the root in X it must be the root of the subtree in Y that spans X and thus an ancestor of all other vertices in X. Therefore it is also an ancestor of all vertices in W with larger turnover time than 2^k. $\square$

Now let v_X be the vertex in X closest to root in G and P_{v_X} a path in G from the root to v_X.

Lemma 5.7. *In any call to $MaxTreeSchedule$ where Y is non-empty, $Y^{anc} \cup Y \cup P_{v_X}$ spans $X^{anc} \cup X$.*

Proof. Let X' be the set of vertices assigned by an ancestor call and $v_{X'}$ the vertex in X' closest to the root in G. As $v_{X'}$ was assigned by an ancestor call we know that $\tau_{v_{X'}} < \tau_{v_X}$. From Lemma 5.6 we then know that $v_{X'}$ is an ancestor of v_X and therefore must lie on P_{v_X}. Since both X' and X are part of connected components in $Y^{anc} \cup Y$ it means that $X' \cup X$ is spanned by $Y^{anc} \cup Y \cup P_{v_X}$. Repeating this argument for all ancestor calls leads to Lemma 5.7. □

We now have a set of edges which we know spans the vertices we are interested in. Now all we need to do is to bound the cost of these edges. Bounding the cost of P_{v_X} is simple as at some point every vertex must be visited. This means that $c(P_{v_X}) \leq \frac{1}{2}OPT^R$. So we now only need to bound the cost of $Y^{anc} \cup Y$.

Lemma 5.8. *Let I^R be an instance of RFTT where G is a tree and all turnover times are powers of two. Then in any call to $MaxTreeSchedule$, we have the following bound*

$$\sum_{Y^{anc}} c(e) + 2^k \sum_{e \in W} \frac{c(e)}{q(e)} \leq \Gamma(I^R)$$

Proof. This we will prove via induction. First we deal with the base case where $k = 0$, $Y^{anc} =$ and W is a DFS traversal of G. In this case we get

$$\sum_{Y^{anc}} c(e) + 2^k \sum_{e \in W} \frac{c(e)}{q(e)} = 2 \sum_{e \in G} \frac{c(e)}{q(e)} = \Gamma(I^R)$$

which satisfies the bound. Now assume the statement holds for depth k. We will now prove that it also holds for depth $k+1$. To do this we need to prove that it holds for both calls made of depth $k+1$. So for $i = 0, 1$, we must prove that

$$\sum_{Y^{anc} \cup Y} c(e) + 2^k \sum_{e \in W_i} \frac{c(e)}{q(e)} \leq \Gamma(I^R)$$

As Y^{anc} and Y are disjoint we can rewrite this to be

$$\sum_{Y^{anc} \cup Y} c(e) + 2^k \sum_{e \in W_i} \frac{c(e)}{q(e)} = \sum_{Y^{anc}} c(e) + \sum_{e \in Y} c(e) + 2^{k+1} \sum_{e \in W_i} \frac{c(e)}{q(e)}$$

As $q(e) = 2^k$ for $e \in Y$ and W_i is part of a well balanced split we then get

$$\sum_{Y^{anc}} c(e) + \sum_{e \in Y} c(e) + 2^{k+1} \sum_{e \in W_i} \frac{c(e)}{q(e)} \leq \sum_{Y^{anc}} c(e) + 2^k \sum_{e \in Y} \frac{c(e)}{q(e)} + 2^{k+1} \left(\frac{1}{2} \sum_{e \in W/Y} \frac{c(e)}{q(e)} \right)$$

Since W/Y is W with the edges Y contracted and some edges of those might appear in W twice, we know that

$$\sum_{e \in W/Y} \frac{c(e)}{q(e)} + \sum_{e \in Y} \frac{c(e)}{q(e)} \leq \sum_{e \in W} \frac{c(e)}{q(e)}$$

With this we finally get

$$\sum_{Y^{anc} \cup Y} c(e) + 2^k \sum_{e \in W_i} \frac{c(e)}{q(e)} \leq \sum_{Y^{anc}} c(e) + 2^k \sum_{e \in Y} \frac{c(e)}{q(e)} + 2^k \left(\sum_{e \in Y} \frac{c(e)}{q(e)} + \sum_{e \in W/Y} \frac{c(e)}{q(e)} \right)$$

$$= \sum_{Y^{anc}} c(e) + 2^k \left(\sum_{e \in Y} \frac{c(e)}{q(e)} + \sum_{e \in W/Y} \frac{c(e)}{q(e)} \right) \leq \sum_{Y^{anc}} c(e) + 2^k \sum_{e \in W} \frac{c(e)}{q(e)} \leq \Gamma(I^R)$$

□

This leads us to the final cost bound for a call to $MaxTreeSchedule$.

Lemma 5.9. *Let I^R be an instance of RFTT where G is a tree and all turnover times are powers of two. Then in any call to $MaxTreeSchedule$, we have the following bound*

$$\sum_{e \in Y^{anc} \cup Y \cup P_{v_X}} c(e) \leq \frac{3}{2} OPT^R$$

Proof. First we get

$$\sum_{e \in Y^{anc} \cup Y \cup P_{v_X}} c(e) \leq \sum_{e \in Y^{anc}} c(e) + 2^k \sum_{e \in Y} \frac{c(e)}{q(e)} + c(P_{v_x})$$

$$\leq \sum_{e \in Y^{anc}} c(e) + 2^k \left(\sum_{e \in W} \frac{c(e)}{q(e)} - \sum_{e \in W/Y} \frac{c(e)}{q(e)} \right) + c(P_{v_x}) \leq \sum_{e \in Y^{anc}} c(e) + 2^k \sum_{e \in W} \frac{c(e)}{q(e)} + c(P_{v_x})$$

By Lemma 5.8 and the fact that the maximum cost of a day must be greater than or equal to the average cost we then get

$$\leq \sum_{e \in Y^{anc}} c(e) + 2^k \sum_{e \in W} \frac{c(e)}{q(e)} + c(P_{v_x}) \leq \Gamma(I^R) + c(P_{v_X}) \leq OPT^R + \frac{1}{2} OPT^R = \frac{3}{2} OPT^R$$

□

With this we are now ready to present the following Lemma for $MaxTreeSchedule$.

Lemma 5.10. *Let I^R be an instance of RFTT where G is a tree and all turnover times are powers of two. Now let W be a DFS walk of G. Then $MaxTree(W, 1, 0)$ is a 3-approximation.*

Proof. **Feasibility** Let S be the solution produces by $MaxTreeSchedule$. Whenever a vertex v is assigned by a call it sets $h(v) = t$ and $p(v) = \tau_v$. This means that in order to prove that v is feasible in S, we need to show that $t \leq \tau_v = 2^k$. This we will do by induction. For the base case we have $t = 1 = 2^0 = 2^k$, so it is true for the base case.
Now assume it holds for depths k. We know need to show that it holds for both child calls of depth $k+1$. Since one of the calls does not change t we will simply prove it for the one that does. In this case we get that $t + 2^k \leq 2^k + 2^k = 2^{k+1}$.
Bound Let r be the route taken on a day where $c(r) = c(S)$. The vertices visited this day correspond to vertices assigned by a leaf call and all of its ancestor calls. For the leaf call it is the set $X^{anc} \cup X$ which by Lemma 5.7 is spanned by $Y^{anc} \cup Y \cup P_{v_X}$. This means that r only uses edges in $Y^{anc} \cup Y \cup P_{v_X}$, however it might use those edges twice. Combining this with Lemma 5.9 we get

$$c(S) = c(r) \leq 2 \left(\sum_{e \in Y^{anc} \cup Y \cup P_{v_X}} c(e) \right) \leq 2 \cdot \frac{3}{2} OPT^R = 3OPT^R$$

Complexity First we note that since every well-balanced split removes at least one edge from W, this means that the number of calls made is at most $2|E|$. Constructing X and assigning $h(v), p(v)$ values can be done in $O(n)$ time, while constructing Y, contracting Y, creating the well-balanced split of W/Y can be done in $O(|E|)$ time. So the running time of $MaxTreeSchedule$ is $O(|E|(n + |E|) = O(n^(n + n)) = O(n^2)$ which is clearly polynomial. Finding the optimal route on a given day is simple for trees so the solution is also polynomial. □

With our analysis of *MaxTreeSchedule* done we can now finally present the following Theorem.

Theorem 5.2. *Algorithm 2 is a 6-approximation for Min-Max on trees.*

Proof. **Feasibility** Let I^R be an instance of RFTT where G is a tree and $I^{\frac{R}{2}}$ be the instance where all turnover times are rounded down to nearest power of two. Now let S be the solution found by *MaxTreeSchedule* to instance $I^{\frac{R}{2}}$. By Lemma 5.10 we know that S is a feasible solution to $I^{\frac{R}{2}}$ which by Lemma 5.1 means it is also a feasible solution to I^R.
Bound Combining Lemma 5.10 and Lemma 5.2 we know that $c(S) \leq 3OPT^{\frac{R}{2}} \leq 6OPT^R$.
Complexity Rounding down all turnover times takes $O(n)$, constructing W and finding $q(e)$ for every $e \in E$ takes $O(n + |E|) = O(n)$ and the call to *MaxTreeSchedule* takes $O(n^2)$ time. So the running time of algorithm 2 is $O(n + n + n^2) = O(n^2)$. Finding the optimal route on a given day is simple for trees, so the solution is polynomial.

□

5.3 Min-Avg on a path

We will now give a pseudo-polynomial algorithm for Min-Avg on a path which is heavily inspired by the one given by [1]. Note that this can also be used to solve Min-Avg on graphs that consists of several paths all rooted at u.

This algorithm uses dynamic programming to solve RFTT on a path. The input is an instance where G is a path rooted at u. As a path is also tree we can assume that the turnover times are non-decreasing along the path. We label the vertices $v_1, v_2 \dots v_n$ with v_1 being the vertex closest to the root and v_n being the vertex furthest from the root. Importantly on a path we know that when v_n is visited, so is every other vertex. This means that every turnover time is reset on the day v_n is visited. So the solution will be repeating the schedule of the first L days, where L is day where v_n is visited. We know that $L \leq \tau_{v_n} = \tau_{max}$ so we will try to find L.
We will use the function $\phi(i, l)$ as our function for calculating the optimal solution. We describe ϕ as follows: $\phi(i, l)$ returns the minimum cost of a schedule that visits vertex v_i on day l. We initialize $\phi(0, l) = 0$ and $\phi(i, 0) = 0$. The recursion of $\phi(i, l)$ we define as follows.

$$\phi(i,l) = \begin{cases} \phi(i-1,l) + d(v_{i-1}, v_i) & \text{if } l \leq \tau_{v_i} \\ \min_{k=1\dots\tau_{v_i}} \phi(i-1,k) + d(v_{i-1}, v_i) + \phi(i, l-k) & \text{if } \tau_{v_i} < l \end{cases}$$

$l \leq \tau_{v_i}$: If $\tau_{v_i} \leq l$ then it is not necessary to visit v_i before it is visited on day l. Therefore the cost is simply $d(v_{i-1}, v_i) + \phi(i-1, l)$ as v_{v_i} is also visited on day l if v_i is.

$\tau_{v_i} < l$: If $\tau_{v_i} > l$ then v must be also be visited on a day before day l. However since we do not know which day we iterate over the days $[l - \tau_{v_i}, l - 1]$. In the case where v_i is visited on day $l - k$, we know that the same is true for v_{i-1}. Therefore the value $\phi(i, l - k)$ will deal with cost of visiting v_{i-1} for the first $l - k$ days. As a result we also need to include the cost for v_{i-1} during the days $[l - k + 1, l]$, which we can do with the value $\phi(v_{i-1}, k)$. Thus the cost becomes $\min_{k=1\dots\tau_{v_i}} \phi(i-1, k-1) + d(v_{i-1}, v_i) + \phi(i, l-k)$.

The Dynamic Program For a given L, the objective value of the solution is $\frac{\phi(n,L)}{L}$. So to find the optimal solution we must find the value $\min_{L\in 1\ldots\tau_{max}} \frac{\phi(n,L)}{L}$. First we calculate every $\phi(i,l)$ value in a table of size $n\tau_{max}$. This table contains all $\phi(n,L)$ values for $L = 1\ldots\tau_{max}$, so we can go through and check $\frac{\phi(n,L)}{L}$ for all these values to find the optimal L. Given the filled out table and the optimal L, there are several ways to find the actual days each vertex is visited. For the sake of brevity we will not cover this, as anyone familiar with dynamic program should be able to do it given the definition of $\phi(i,l)$. We will only note that it is clearly possible to find a solution in the same time it takes to construct the table.

Complexity To fill out the table we need to calculate $n\tau_{max}$ values. When calculating a value $\phi(i,l)$ we might need to compare $\tau_{v_i} \leq \tau_{max}$ different values. So calculating a single entry in the table takes $O(\tau_{max})$ time. It therefore takes $O(n\tau_{max}^2)$ time to fill out the table. Then finding the optimal value for L means comparing τ_{max} values and thus takes $O(\tau_{max})$ time. So in total the time complexity of finding L is $O(n\tau_{max}^2)$, which is an improvement to the τ_{max}^3 time complexity of the dynamic program given by [1].

All of this lets us state the following Theorem.

Theorem 5.3. *Min-Avg can be solved on a path in $O(n\tau_{max}^2)$ time.*

6 RFTT on Cycles

In this chapter we will give several new results. For Min-Max we will give a polynomial time algorithm on cycles. For Min-Avg we will give insight into how a potential pseudo-algorithm can be achieved on cycles. Additionally we will give a 4-approximation for Min-Avg on cycles.
We will begin by defining some notation for cycles. First we define the cost of traversing a cycle.

Definition 6.1. *Let I^R be an instance of RFTT where G is a cycle. Then the cost of traversing the entire cycle is $C_G = \sum_{e \in E} c(e)$.*

Next we define some important vertices and paths on a cycle.

Definition 6.2. *Let I^R be an instance of RFTT where G is a cycle. Then a, b are the two vertices furthest from the root. Furthermore P_a, P_b are the shortest paths in G from the root to a and b respectively. The edge between a and b we denote δ. Let a, b be chosen such that $c(P_a) \geq c(P_b)$. Note that $c(P_b) \leq c(P_a) \leq \frac{C_G}{2}$, $c(P_a) + c(\delta) \geq \frac{C_G}{2}$ and $c(P_b) + c(\delta) \geq \frac{C_G}{2}$.*

The situation described in definition 6.2 is shown in figure 6.1.

6.1

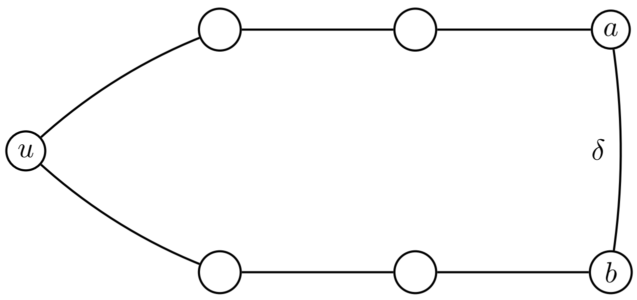

Figure 6.1: Example of the vertices a, b and paths P_a, P_b on a cycle.

With this we can now present the following Lemma.

Lemma 6.1. *Let I^R be an instance of RFTT where G is a cycle. Then without wlog. we may assume that turnover times along P_a, P_b are non-decreasing.*

Proof. We will wlog. prove this for P_a. Take vertices v, w on P_a where v is closer to the root than w and $\tau_v > \tau_w$. We know that w must be visited at least every τ_w days. There are two possible ways this can happens. Either w is visited directly via edges in P_a in which case v is also visited or w is visited by first traversing P_b and then the shortest path from b to w. In the second case we have already payed a cost of at least $c(P_b) + c(\delta) \geq \frac{C_G}{2}$, so going back via the same path will bring the total cost to at least C_G. It will therefore always be optimal to simply go from w to the root, in which case v is also visited. □

In this section we also need to find optimal routes on cycles. This is quite simple, so we wont provide a detailed algorithm, but only give the following Lemma.

Lemma 6.2. *Let I^R be an instance of RFTT where G is a cycle. Now let V_s be a subset of V. Then it is possible to find an optimal route that visits V_s in $O(n)$ time.*

Proof. Let a', b' the vertices in V_s on P_a, P_b that are furthest from the root. Any route that visits these will visit all vertices in V_s. Let $P_{a'}, P_{b'}$ be shortest routes from the root to a' and b' respectively. There are two possible routes that visit both a' and b'. The one that traverses the entire cycle and the one traverses $P_{a'}$ and $P_{b'}$ twice. So the optimal is simply the cheapest of those two. Calculating the cost of two routes and comparing can easily be done in $O(n)$ time. □

6.1 Min-Max solved on a cycle

We can now give the algorithm to Min-Max on the cycle.

Algorithm 4

```
if $\tau_a = 1$ or $\tau_b = 1$ then
    Let $h(v) = p(v) = 1$ for all $v \in V$
    Return $h(v)$ and $p(v)$
end if
For all $v \in \{v \mid v \in V \wedge \tau_v = 1\}$ set $h(v) = p(v) = 1$
For the remaining vertices on $P_a$ set $h(v) = 1$ and $p(v) = 2$
For the remaining vertices on $P_b$ set $h(v) = 2$ and $p(v) = 2$
Return $h(v)$ and $p(v)$
```

Which lets us present the following Theorem.

Theorem 6.1. *Algorithm 4 finds an optimal polynomial solution for Min-Max on cycles in $O(n)$ time.*

Proof. **Feasibility** . Let I^R an instance of RFTT where G is a cycle and S be the solution produced by algorithm 4. Any vertex v with $\tau_v = 1$ has $h(v) = p(v) = 1$ and any other vertex either has $h(v) = p(v) = 1$ or $h(v) \leq 2$ and $p(v) = 2$, so clearly S is feasible.
Optimality If $\tau_a = 1$ then at some point we must visit both a and b on the same day. On this day we visit all vertices in G. Since this is the worst possible outcome for Min-Max, nothing we do on any other day can be worse, so for simplicity we visit every vertex every day. The argument for $\tau_b = 1$ is symmetrical.
Now assume that $\tau_a > 1$ and $\tau_b > 1$. At some point we must visit a. On this day we must also visit any vertices on P_b with turnover time 1. Let the cost of the optimal route visiting these vertices be r_a. We can make the same argument for b and label the corresponding route r_b. The cost of the maximum route in any solution to I^R must be at least $\max(c(r_a), c(r_b))$. Since r_a and r_b are the only two routes ever taken by S, we have that $c(S) = \max(c(r_a), c(r_b)) = OPT^R$.
Complexity Assigning the $h(v)$ and $p(v)$ values for all the vertices can easily be done in $O(n)$ time, so this is clearly the time complexity of algorithm 4. From Lemma 6.2 we know we can find the optimal route on a given day in $O(n)$ time. So S is a polynomial solution. □

6.2 Min-Avg on Cycles

6.2.1 Pseudo-polynomial Algorithm

In this section we will give insights into how a potential pseudo-polynomial algorithm for Min-Avg on cycles can be achieved. The idea for this algorithm is very similar to the pseudo-polynomial algorithm for Min-Avg on paths. On paths we now that all turnover times are reset when the vertex furthest from the root is visited. For cycles we know the same when the entire cycle is visited. We will show that if we know when an optimal solution visits the entire cycle, we can find a solution in pseudo-polynomial time. First we define some more vertices and paths on cycles.

Definition 6.3. *Let I^R be an instance of RFTT where G is a cycle. Now let L be the day where an optimal solution visits the entire cycle. Then a_L, b_L are the vertices on P_a, P_B furthest from the root with turnover time less than L. Let P_{a_L}, P_{b_L} be subpaths of P_a, P_b that end in a_L and b_L.*

With this definition we are ready to present the following Lemma.

Lemma 6.3. *Let I^R be an instance of RFTT where G is a cycle. For a given day L, the optimal solution to I^R, that visits all vertices in G on day L, can be found in $O(nL\tau_{max})$ time.*

Proof. Every vertex not on P_{a_L} or P_{b_L} must have a turnover time of at least L. As we visit the entire cycle every L days, all of these vertices are feasible. We still have to provide a feasible schedule for P_{a_L} and P_{b_L}. Since we know L, we know that the optimal solution does traverse the entire cycle during the first $L-1$ days. This means that we can treat P_{a_L}, P_{b_L} as two separate paths. This leaves us in the following situation. We have a path, P_{a_L} where we know the day, L, where the vertex furthest from the root, a_L, is visited. This is exactly what the pseudo-polynomial algorithm on paths given earlier solves. Now in this case $\tau_{a_L} < L$, however there is nothing in how we defined $\phi(i,l)$ that prevents this application of the algorithm. In fact it is very important that $\phi(i,l)$ works in these instances, as it otherwise would not work at all.
To find the solution to P_{a_L} we must fill out a table of size $O(nL)$. The time it takes to calculate a single entry does not depend on the size of the time, but the maximum turnover time, so to fill out the table takes $O(nL\tau_{max})$ time. Finding the solution to P_{b_L} naturally takes the same amount of time. □

This is a nice result, but still leaves us with one problem. For paths, bounding L was quite easy, as the vertex furthest from the root must be visited within the first τ_{max} days. Bounding L for cycles is quite a lot harder. First note that it is possible that $\tau_{max} < L$. Imagine the following example. I^R is an instance where G is a cycle. In G it holds that $C_G > 2(c(P_a) + c(P_b)$. In this case it is clear that an optimal solution never uses δ. So the solution is to treat P_a and P_b as two separate paths. Let L_a, L_b be the days that a, b are visited for the first time in an optimal solution. Since the schedules for P_a, P_b simply repeat, we know that every vertex in the cycle is visited on day $L_a \cdot L_b$. So for this instance, $L = L_a \cdot L_b = O(\tau_{max}^2$. In the case where $C_g \leq 2(c(P_a) + c(P_b)$ it is harder to bound L. We do believe it to be possible, even if we did not manage it in this thesis, so we will present the following Theorem.

Theorem 6.2. *If for all instances of RFTT, where G is a cycle, L is bounded by some polynomial $P(\tau_{max})$, then Min-Avg can be solved on the cycle in $O(nP(\tau_{max})^2\tau_{max})$.*

Proof. Let I^R be an instance of RFTT where G is a cycle. For each $l = 1 \dots P(\tau_{max})$ use Lemma 6.3 to find the best solution for that l in $O(nl\tau_{max})$ time. Then simply choose the best solution among these. Since $L \leq P(\tau_{max})$ we know that $l = L$ at some point. The optimal solution must therefore be among the all the solutions calculated. In total this takes $O(\sum_{l=1}^{P(\tau_{max})} O(nl\tau_{max}) = O(nP(\tau_{max})^2\tau_{max})$. □

6.2.2 4-approximation algorithm

We will now give a 4-approximation algorithm for Min-Avg on the cycle. We will in this algorithm again round down every turnover time to nearest power of two. We start by providing some relevant Lemmas for such instances. First we show that the optimal schedule for P_{a_L} and P_{b_L} is always predictable given L.

Lemma 6.4. *Let I^R be an instance of RFTT where G is a cycle and all turnover times are powers of two. Now let L be the day where the entire cycle is traversed. Then the optimal solution for P_{a_L} and P_{b_L} during the first $L-1$ days, will be to visit vertex v on days where $t \equiv 0 \mod \tau_v$.*

Proof. This we can prove by calculating a lower bound for the average and show that this is what we achieve. Given the day L, we need to create a schedule for P_{a_L} for the first $L-1$ days. We can lower bound the average cost for these $L-1$ days as follows

$$\frac{2}{L-1}\sum_{e\in P_{a_L}} c(e)\cdot\left\lfloor\frac{L-1}{q(e)}\right\rfloor$$

This bound can be achieved by following the same logic as applied in the proof of Lemma 5.3. This value is exactly achieved if we visit every vertex as specified. □

We next show that L is bounded when turnover times are all a power of two. To do this we first introduce some notation. We look at the optimal schedule where the entire cycle is never traversed. This is just the optimal path solutions for P_{a_L} and P_{b_L}. Since the turnover times are powers of two these solutions are perfectly synchronized, meaning that when all of the vertices with turnover time 4 are visited on P_{a_L} it also happens on P_{b_L}. As a result we only have k unique routes in our solution where k is the number of different powers of two among the turnover times. Every route corresponds to a power of two and is taken on every day that is a multiple of that power of two. We will label the routes by their corresponding power of two. Since every route also visits all vertices visited by routes with smaller corresponding powers of two we can list these routes in non-decreasing order. Let $r_1, r_2, r_4 \dots r_{\tau_{max}}$ be the routes labeled and listed as just described. Among these let r_f be the cheapest route where $c(r_f) > C_G$. Now we can present the following Lemma.

Lemma 6.5. *Let I^R be an instance of RFTT where G is a cycle and all turnover times are powers of two. Now let r_f be the cheapest route where $c(r_f) > C_G$. Then L, the day where the entire cycle is traversed by an optimal solution, cannot be larger than f. If r_f does not exist then there exists an optimal solution that never traverses the entire cycle.*

Proof. We first deal with the case where r_f does not exist. Since $r_{\tau_{max}}$ must visit all vertices, traversing the cycle is not the cheapest way of visiting all the vertices. This means that any solution that traverses the entire cycle can always be made into on that never does not by never using δ without increasing the cost. So there will always exist an optimal solution that never traverses the entire cycle.
Now we deal with the case where r_f exists. Assume L is greater than f. We now apply Lemma 6.4 and find the solution for P_{a_L} and P_{b_L}. Since $L > f$ we on day f visit all vertices with turnover times that divide f. This is exactly the route r_f. Since we know that $c(r_f) > C_G$ we can make this solution cheaper by traversing the entire cycle on day f as well. This means that any solution with $L > f$ can be made cheaper and thus cannot be optimal. □

Now that we have bounded L, we provide a Lemma which will narrow down the options for L even further.

Lemma 6.6. *Let I^R be an instance of RFTT where G is a cycle and all turnover times are powers of two. Now let S be a solution to I^R with $L = i$ with i being some power of two less than f. Then it is always possible to create a solution S' with $L' = 2i$, such that $c(S') \leq c(S)$.*

Proof. Look at the first $2i$ days of both solutions. The only difference between the schedules of these two solutions is on day i. All the other days have the exact same costs and as a result we only need to compare the routes they take on day i. On day i, solution S traverses the entire cycle while solution S' uses route r_i. Since $i < f$ we know that $c(r_i) \leq C_G$ so the cost of S' cannot be larger than the cost of S. □

With this we will now present an algorithm for approximating Min-Avg on cycles.

Algorithm 5

```
Round down every turnover time to nearest power of two.
Calculate the routes $r_1, r_2, r_4, \dots, r_{\tau_{max}}$ and their costs.
**if** $r_f$ does not exists **then**
    For all $v \in V$ set $h(v) = p(v) = \tau_v$
    Return $h(v)$ and $p(v)$
**end if**
Set $L = f$
For all $v$ with $\tau_v \geq f$ set $h(v) = p(v) = f$
For all $v$ with $\tau_v < f$ set $h(v) = p(v) = \tau_v$
Return $h(v)$ and $p(v)$
```

Which leads us to presenting the following Theorem.

Theorem 6.3. *Algorithm 5 is a 4-approximation for Min-Avg on a cycle.*

Proof. **Feasibility** Let I^R be an instance of RFTT where G is a cycle and let $I^{\frac{R}{2}}$ be the instance where turnover times have been rounded down. Now let S be the solution produced by algorithm 4 to $I^{\frac{R}{2}}$.
If r_f does not exist, then every vertex v is assigned the values $h(v) = p(v) = \tau_v$ and is clearly feasible. If r_f does exist, then every vertex v is either assigned the values $h(v) = p(v) = \tau_v$ or $h(v) = p(v) = f$. In either case v is feasible. Combining this with Lemma 5.1 we get that S is a feasible solution to I^R.

Bound Let S' to be an optimal solution to $I^{\frac{R}{2}}$ where $c(S') = OPT^{\frac{R}{2}}$ and $L' = i$. Now let S'' the be cheapest solution with $L'' = 2^{\lfloor \log(i) \rfloor}$. Let C_i be the cumulative cost of the first $2^{\lfloor \log(i) \rfloor} - 1$ routes in S'. From Lemma 6.4 we know that the routes for S' and S'' during these days are the same. This means that C_i is also the cost of the first $2^{\lfloor \log(i) \rfloor} - 1$ routes in S''. As both solutions also traverse the entire cycle, we get that $c(S'') = \frac{C_i + C_g}{2^{\lfloor \log(i) \rfloor}}$ and that $\frac{C_i + C_G}{i} \leq c(S')$. Since we know that $2^{\lfloor \log(i) \rfloor} \geq \frac{i}{2}$ we get the following relation.

$$c(S'') = \frac{C_i + C_g}{2^{\lfloor \log(i) \rfloor}} \leq \frac{C_i + C_g}{\frac{i}{2}} = 2\frac{C_i + C_g}{i} \leq 2 \cdot c(S') = 2OPT^{\frac{R}{2}}$$

So we get that $c(S'') \leq 2OPT^{\frac{R}{2}}$. We can now apply Lemma 6.6 until $L'' = f$. This final solution is S. Combining all of this with Lemma 5.2 we get

$$c(S) \leq c(S'') \leq 2OPT^{\frac{R}{2}} \leq 4OPT^{R}$$

Complexity Rounding down the turnover times takes $O(n)$ time. From Lemma 6.2 we know that calculating the cost of a single route takes $O(n)$ time. As there are at most n different powers of two among the turnover times, we at most need to calculate the cost of n routes. So calculating the route costs takes $O(n^2)$ time. Assigning the $h(v), p(v)$ values takes $O(n)$ time. Therefore the algorithm takes $O(n^2)$ time and by Lemma 6.2 the solution found is polynomial. $\square$

7 Trees with added cycles

We will now use the result for cycles and trees to give approximation algorithms for a new family of graphs. These graphs will be combinations of cycles and trees. First we define this new family of graphs.

Definition 7.1. *A graph G where all cycles are disjoint is called a cycle-tree.*

An example of a cycle-tree can be seen in figure 7.1.

We now define some important vertices in cycle-trees.

Definition 7.2. *Let I^R be an instance of RFTT where G is a cycle-tree. Now let C be a cycle in G. Then u_C is the vertex on C closest to the root of G. We call u_C the root of C..*

Next we define an important operation on cycle trees.

Definition 7.3. *Let I^R be an instance of RFTT where G is a cycle-tree and*

is cycle in G. Then contracting C corresponds to contracting every edge in C. The resulting vertex has turnover time τ_{u_C}.

To ensure that the solution produced by the algorithms, we give, are polynomial, we give an algorithm to find an optimal route given a subset of vertices to visit. Such an algorithm takes as input an instance, I^R, of RFTT where G is a cycle-tree and a subset of vertices, V_S, finds an optimal route, that visits V_s.

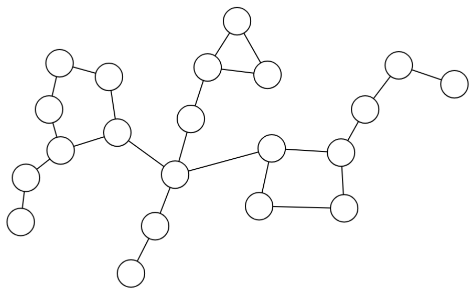

Figure 7.1: Example of a cycle-tree

Algorithm 6

```
Mark all vertices in $V_S$
Find all cycles in $G$.
for $v \in V_S$ do
    Let $P$ be a path from the root to $v$.
    Mark all vertices on $P$ not in a cycle.
    Mark all vertices on $P$ that are connected to a marked vertex.
end for
Let $G_T$ be $G$ where all cycles have been contracted.
Find the optimal route, $r$, visiting all remaining marked vertices in $G_T$.
for every cycle $C$ in $G$ do
    Find the optimal route, $r_C$, on $C$ that visits all marked vertices in $C$ from $u_C$.
    Uncontract $C$ in $G_T$.
    Modify $r$ in the following way:
    Follow $r$ until it reaches $u_C$. Then follow $r_C$ until you reach a vertex, $v$, which is
    incident to a vertex, $v_r$, that is in $r$. The edge between $v$ and $v_r$ was in $G_T$ and therefore
    in $r$. Now follow $r$ until it reaches $v$ again. Do this until $r_C$ reaches $u_C$ again and then
    follow $r$ back to $u$.
end for
Return $r$.
```

Which leads us to presenting the following Theorem.

Theorem 7.1. *Let I^R be an instance of RFTT where G is a cycle-tree. Now let V_S be a subset of V. Then algorithm* 6 *finds an optimal route that visits V_S in $O(n^2)$*

Proof. **Feasibility** Initially r visits all vertices in V_S that are not in a cycle in G. Then for each cycle, C, r is updated so it also visits the marked vertices of C. As these include the vertices of V_S that are in C, we can be sure that r visits all vertices in V_S.
Optimality Let v be a vertex in V_S and P a path from the root to v. Now let e be an edge in P that is not in a cycle in G. Then e must be in every path from the root to v. To prove this assume there exists a path P' from the root to v that does not use e. Let v_1, v_2 be the endpoints of e where v_1 is the closest to the root. Now follow P from v_2 towards v until a vertex v_3, which is also on P', is reached. Label this path α. From v_3 follow P' towards the root until a vertex v_4, which is also on P, is reached. Label this path β. Finally from v_4 follow P towards v until v_1 is reached. Label this path γ. This construction is illustrated in figure 7.2. Since P and P' are both root-to-v paths we know that they meet in u and in v. This means that α, β and γ are all valid constructions. Since e is not in a cycle, it not possible to reach v_2 without using e. This means that cannot v_2 be on P', and α contains at least one edge. By construction none of the vertices on α, apart from v_3, are on P', so β must also contains at least one edge and v_4 must either be v_1 or a vertex on P closer to the root. This means that γ might contain no edges, but this is fine. By construction the paths α, β, γ share no edges and do not contain e. So we look at the following construction

$$v_2 \to \alpha \to \beta \to \gamma \to e \to v_2$$

This is a cycle in G containing e, but we know no such cycle exists. Therefore P' does not exists. This also means that v_1 and v_2 are on every root-to-v path.
From this we can also conclude that it is not possible to visit v and return to the root without traversing e twice. As a result we know that every edge not in a cycle but on a

path to some vertex in V_s must appear twice in any route starting in u that visits all the vertices in V_s. This is exactly the case for the initial r.
Now let C be a cycle in G and V_r the set of vertices in C that we cannot avoid visiting. V_r might contain vertices that are incident to edges, that are not in any cycle, vertices that are in V_s or both. Let $v \neq u_C$ be a vertex in V_r and P_l, P_r be the two paths from u_C to v. As all cycles are disjoint, any root-to-v path must contain either P_l or P_r. The same goes for any v-to-root path. This means that any route starting in u, that visits all vertices in V_r, must contain edges that correspond to a route in C. Now let r_C be the optimal route in C, that visit all of V_r. Then in any route, that visits all of V_r, the cost of the edges in the route, that are also in C, is at least $c(r_C)$.
This means that the cost of any route, that visits all of V_s, is at least

$$\sum_{e \in E_{V_s}} 2 \cdot c(e) + \sum_{C \in CG} c(r_C)$$

$$\text{where } E_{V_s} = \{e \mid e \in E \wedge e \text{ is on a path to a vertex in } V_s \wedge e \text{ is not in any cycle}\}$$
$$\text{and } CG = \text{The set of all cycles in } G.$$

Which is exactly the cost of r when it is returned by algorithm 6.
Complexity Marking all vertices in V_s takes $O(n)$ time. Then for every vertex in V_s, we find a path from the root to v and mark vertices on the path. Finding the path and marking the relevant vertices on it can be done in $O(n)$ time. So doing this for all vertices in V_s takes $O(n^2)$ time.
As all cycles are disjoint, detecting and finding all the cycles can be done with DFS in $O(n)$ time. Contracting them all also takes $O(n)$ time and so does finding the initial r in G_T. Then for every cycle, C, we do the following. Find an optimal route that visits every marked vertex in C, which by Lemma 6.2 takes $O(n)$ time. Then to uncontract C takes $O(n)$ time. To modify r we will traverse both the entirety of r and r_C which takes $O(n)$ time. As there cannot be more than n cycles, this step takes $O(n^2)$ in total. Therefore the algorithm has a time complexity of $O(n^2)$. □

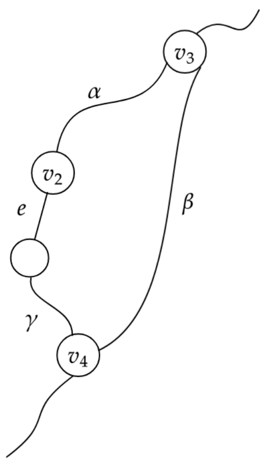

Figure 7.2: The construction described in the proof of Theorem 7.1

.

Now that we have an algorithm for calculating optimal routes in cycle-trees, we will provide two Lemmas on properties of turnover times in cycle-trees.

Lemma 7.1. *Let I^R be a instance of RFTT where G is cycle-tree. Now let C be a cycle in G. Then we may assume that turnover times are non-decreasing along P_a and P_b.*

Proof. We will only prove this for P_a, as the argument for P_b is symmetric. Let $v \neq u_C, w \neq u_C$ be vertices on P_a, where v is furthest from u_C. As cycles are disjoint, v cannot be visited without visiting u_C. From there any route, starting in u, visiting v must either use some the edges on P_a to v or all edges on P_b, then δ and then the shortest from path from a to v. In the first case w is also visited. In the second case, the cost of visiting v is at least $c(P_b) + c(\delta) \geq \frac{C_G}{2}$. As the route must return to u it must go from v back to u_C again. To do this it must either use the edges in C is has already used or it can use the edges on P_a from v to u_C. The first option costs at least $2(c(P_b) + c(\delta)) \geq C_G$ where as the second option costs exactly C_g as it contains all the edges in C once. This means that we can turn any solution that does not visit w whenever it visits v into one that does, without increasing the cost of the solution. Thus we may assume that every vertex on P_a closer to u_C than v, is visited at least as often as v. □

Next is a similar Lemma for vertices not on a cycle.

Lemma 7.2. *Let I^R be a instance of RFTT where G is cycle-tree. Now let P be the root-to-vertex path to some vertex w in G. Furthermore let v be a non-root vertex on P which is either the root of a cycle or not in a cycle. Then we may assume that $\tau_v \leq \tau_v$.*

Proof. Since v is either the root of a cycle or not in any cycle, it must be on all paths from the root to w. Therefore it is not possible to visit w without visiting v. □

Finally we define a property on edges that do not appear in a cycle.

Definition 7.4. *Let I^R be a instance of RFTT where G is cycle-tree. Now let e be an edge not in a cycle and v the vertex incident to e furthest from the root. Then $q(e) = \tau_v$.*

7.1 Approximating Min-Max on cycle-trees

We are now ready to give an approximation algorithm for Min-Max on cycle-trees.

Algorithm 7

```
  Find every cycle $C_1, C_2, \dots$
  For every vertex $v$ in a cycle or on a path to a cycle let $h(v) = p(v) = 1$
  Let $G_T$ be the tree where each cycle has been contracted to single vertex.
  Use algorithm 2 to compute $h(v)_T$ and $p(v)_T$ on $G_T$.
  For every vertex $v$ not in a cycle or on a path to a cycle let $h(v) = h(v)_T$ and $p(v) = p(v)_T$
  return $h(v)$ and $p(v)$
```

This leads us to presenting the following Theorem.

Theorem 7.2. *Let N_C describe the number cycles in a graph. Algorithm 7 is a $6 + 2N_C$-approximation algorithm for Min-Max on cycle-trees.*

Proof. **Correctness** Let I^R be an instance of RFTT and S the solution created by algorithm 7. Every vertex on a cycle or on a path to a cycle is visited every day so all of those vertices are feasible in S. Every other vertex must be in G_T. Let S_T be the solution algorithm 2 creates for G_T, which by Theorem 5.2 we know to be feasible. Visiting all the vertices not on a cycle or path to a cycle the same days as in S_T must mean they are also feasible in S. Therefore all vertices are feasible in S and thus S is feasible.

Bound First we bound the cost of visiting all the cycles. Let C be a cycle in G. First we must visit u_c. This can be done using edges with a total cost of $d(u, u_C)$. Next we must visit all the vertices on C and return to u_C. To do this we can use all the edges on both P_a and P_b twice. Then we must return from u_C and go to u, which again costs $d(u_C)$. The cost of visiting C and all vertices on the path to u_C is then at most $2(d(u, u_C)+c(P_a)+c(P_b))$. Since $d(u, u_C)+c(P_a) = d(u, u_C)+d(u_C, a) = d(u, a) \leq \frac{1}{2}OPT^R$ we get the that $2(d(u, u_C)+c(P_A)+c(P_b)) \leq 2(d(u, u_C)+d(u_C, a))+2(d(u, u_C)+d(u_C, b)) \leq OPT^R + OPT^R = 2OPT^R$. As a result every cycle in G adds two to the approximation factor.

Now let us look at a vertex v not in a cycle or on a path to a cycle. The path from u to v in G_T is either identical to the path in G or it crosses a vertex which corresponds to contracted cycle. If it is identical then we can visit v using only the edges used by S_T. If the paths are different it means that on the path to v in G_T we visit at least one vertex that corresponds to a cycle in G. Let P_{vT} be the path to v in G_T and let v_c be the vertex corresponding to the contracted cycle on P_{vT} closest to v. Now let C be the cycle contracted into v_c and w_c the vertex on C closest to v. This situation is illustrated in figure 7.3. Since w_c is visited every day we have already accounted for the cost of visiting it. This means that we can visit v where the only additional edges used are ones used by S_T. We now get the bound $c(S) \leq c(S_T) + 2N_C \cdot OPT^R$. As contracting edges never increases the cost of a solution we get that $OPT^T \leq OPT^R$, where OPT^T is the optimal value for G_T. With this and Theorem 5.2 we get $c(S) \leq 6OPT^T + 2N_C \cdot OPT^R \leq 6OPT^R + 2N_C \cdot OPT^R = (6 + 2N_C)OPT^R$.

Complexity Finding all cycles in a graph, where all cycles are disjoint, can be done using BFS, so this takes $O(n)$ time. We know that algorithm 2 is polynomial so every step of algorithm 7 is polynomial and must therefore also take polynomial time. On a given day we can use algorithm 6 to find the optimal route in polynomial time, so the solution is also polynomial. □

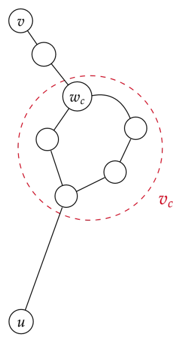

Figure 7.3: Illustration of the situation described in the proof of Theorem 7.2.

7.2 Approximating Min-Avg on cycle-trees

We will now give an approximation algorithm on for Min-Avg on cycle-trees. First we provide a lower bound for the objective value of Min-Avg on cycle-trees

Lemma 7.3. *Let I^R be an instance of RFTT where G is cycle-tree. Now let $C_1, C_2, \dots C_{N_C}$ be the cycles in G and E_C the edges in G that appear in a cycle. Then a lower bound for an optimal solution for Min-Avg is*

$$OPT^R \geq \left(2 \sum_{e \in E \setminus E_C} \frac{c(e)}{q(e)}\right) + \sum_{i=1}^{N_C} OPT^{C_i}$$

Proof. Let e be an in edge in E/E_C. Now let v_1, v_2, be the vertices incident to e, such that v_2 is further from the root. As e is not in a cycle, it must be used at least every $\tau_{v_2} = q(e)$ days. Furthermore, any route beginning in u that uses e, must use e at least twice in order to return to u. Now we apply the same logic as in Lemma 7.3 and get that

$$OPT^R \geq 2 \sum_{e \in E \setminus E_C} \frac{c(e)}{q(e)}$$

Now let S be an optimal solution to I^R and r_t the route taken by S on day t. Furthermore let C be a cycle in G and contract every edge in r_t not in C. As all cycles are disjoint no edge between any to vertices in C have been contracted. Via the argument presented in the proof of Lemma 6 we know that r_t now corresponds to a route in C. Now repeat this process for all routes in S. The result is a solution S' to C. Since every vertex v is visited on the same days in S as it is in S', we know that S' is feasible. It is therefore clear that $c(S') \geq OPT^C$. This means that the cost the edges of C contribute to the average must be at least OPT_C. With this we arrive at the lower bound presented by the Lemma. □

Now we are ready to give the approximation algorithm for Min-Avg on cycle-trees

Algorithm 8

```
  Round down all turnover times to nearest power of two.
  For every $v$, where $v$ either is root of a cycle or not in a cycle, let $h(v) = p(v) = \tau_v$.
  For every cycle $C$ in $G$ use algorithm 5 to find $h(v), p(v)$ values for the non-root vertices
  in $C$.
  Return $h(v)$ and $p(v)$.
```

Which leads us to presenting the following Theorem.

Theorem 7.3. *Algorithm 8 is a 4-approximation for Min-Avg on cycle-trees.*

Proof. **Feasibility** Let I^R be an instance of RFTT and $I^{\frac{R}{2}}$ the instance where the turnover times have been rounded down. Now let S be the solution to $I^{\frac{R}{2}}$ produced by algorithm 7. For every vertex not in a cycle and every root of a cycle, we have that $h(v) = p(v) = \tau_v$, so those vertices are feasible in S. The remaining vertices have $h(v), p(v)$ values assigned by algorithm 5, which by Theorem 6.1 means they are also feasible. So every vertex is feasible in S, which combined with Lemma 5.1 means that S is a feasible solution to I^R.

Bound First we bound the cost of the edges not in a cycle. Let e be an edge not in a cycle. Now let v_1, v_2 be the vertices incident to e, such that v_2 is further from the root.

As $h(v_2) = p(v_2) = \tau_{v_2} = q(e)$ we know that we from e incur a cost of at least $2\frac{c(e)}{q(e)}$. The only way we incur more than that is if e is ever used to visit a vertex with turnover time smaller than τ_{v_2}. Assume there is vertex w that cannot be visited without using e, where $\tau_w < \tau_{v_2}$. Now let P be a path from the root to w. By assumption this contains v, so from Lemma 7.2 we then know that $\tau_{v_2} \leq \tau_w$. This is a contradiction and thus w cannot exist. This means that e is only ever used on days t where $t \equiv 0 \mod q(e)$. Thus we from e incur exactly the cost $2\frac{c(e)}{q(e)}$.
Next we bound the cost of the edges in a cycle. Let C be a cycle in G and S' the solution to C found by algorithm 5. We know that S visits the same vertices on day t as S'. Furthermore we know that the route taken by S on day t uses same edges to visit these vertices as S'. This means that the cost incurred by C is exactly $c(S')$. From Theorem 6.3 we know that $c(S') \leq 4OPT^C$, however as the turnover times are already rounded down to nearest power of two, we can improve this to $c(S') \leq 2OPT^C$. Applying this to every cycle we can using Lemma 5.2 bound the cost of S.

$$c(S) \leq \left(2\sum_{e\in E\setminus E_C}\frac{c(e)}{q(e)}\right) + \sum_{i=1}^{N_C} 2OPT^{C_i} \leq 2\left(\left(2\sum_{e\in E\setminus E_C}\frac{c(e)}{q(e)}\right) + \sum_{i=1}^{N_C} OPT^{C_i}\right)$$
$$\leq 2OPT^{\frac{R}{2}} \leq 4OPT^R$$

Complexity Rounding down every turnover time and assigning $h(v), p(v)$ values for the vertices, that are either the root of a cycle or not in any cycle, can easily be done in $O(n)$ time. From theorem 6.3 we know that algorithm 5 produces a solution in polynomial time. As there at most n cycles, finding $h(v), p(v)$ values for the in cycles vertices also takes polynomial time.
Finding the optimal route on a given day takes $O(n^2)$ time by Theorem **??**. So S is polynomial and found in polynomial time. □

8 Future Work

The material presented in this thesis provides plenty of opportunities for further work. Not only from the new results and algorithms given, but also the material from [1] that we covered. Specifically we note that the 6-approximation for Min-Max on trees given by [1], does not yet have a tight example showing that the analysis cannot be improved. In our attempts it has proven difficult to produce an example, that shows it is worse than a 4-approximation. It would be interesting to see if it is actually a 6-approximation or the analysis could be improved. Among our own results there are many possible improvements. The first and most obvious one would be to bound L for all instances of Min-Avg on cycles. With the results we provided, this would immediately result in a pseudo-polynomial algorithm for Min-Avg on cycles. As mentioned we believe it should be possible to bound L, especially with the insights we have provided, so this provides a clear opportunity for further work.

More work could also be done on our 4-approximation for Min-Avg on cycles. We ask if it is possible to show that L is always a power of two in these cases? If this is true then both algorithm 5 and 8 become 2-approximations. If it proves not be the case, it would imply that there exists a tight example showing that algorithm 5 is not a 2-approximation. It would also be interesting to see how hard it is to extend algorithm 5 to cycles with one or more chords, and what effect it has on the approximation factor. This might also lead to constant factor approximation algorithms for more complex graphs than cycle-trees and help towards finding a constant factor approximation algorithm for general graphs.

Finally we believe that with more time it should be possible to improve our simple and quite poor approximation algorithm for Min-Max on cycle-trees. The goal would be to improve it to a constant factor approximation, that does not depend on the number of cycles in the graph.

The study of RFTT is still relatively young, so there are many opportunities for future work, but the ones we have just described all provide a clear path forward towards tangible results.

9 Conclusion

In this thesis, we have studied Replenishment Problem with Fixed Turnover Times (RFTT), first defined in [1], under both Min-Avg and Min-Max metrics. For Min-Max RFTT we have covered the 6-approximation for trees given by [1], given a $O(n)$ algorithm for cycles and a simple $6 + N_C$ approximation for cycle-trees. Here N_C denotes the number cycles in a graph, and a cycle-tree is a graph where all cycles are disjoint. For Min-Avg we covered the 2-approximation for trees and gave a 4-approximation for cycles. These we combined to also give a 4-approximation for cycle-trees. Additionally we gave insight into how a pseudo-polynomial algorithm on cycles might be achieved and gave an algorithm for finding optimal routes on cycle-trees. Finally on the basis of our result we pointed out several clear paths towards achieving new results and improving the ones presented in this thesis.

Bibliography

[1] Bosman et al. “Approximation Algorithms for Replenishment Problems with Fixed Turnover Times”. In: *Algorithmica* (2022).

[2] Bosman T. N. “Relax, Round, Reformulate: Near-Optimal Algorithms for Planning Problems in Network Design and Scheduling.” In: *PhD-Thesis - Research and graduation internal, Vrije Universiteit Amsterdam* (2019).