Approximation Algorithms for the Traveling Repairman and Speeding Deliveryman Problems

Greg N. Frederickson · Barry Wittman

Received: 28 May 2009 / Accepted: 7 April 2011 / Published online: 22 April 2011
© Springer Science+Business Media, LLC 2011

Abstract Constant-factor, polynomial-time approximation algorithms are presented for two variations of the traveling salesman problem with time windows. In the first variation, the traveling repairman problem, the goal is to find a path that visits the maximum possible number of locations during their time windows. In the second variation, the speeding deliveryman problem, the goal is to find a path that uses the minimum possible speedup to visit all locations during their time windows. For both variations, the time windows are of unit length, and the distance metric is based on a weighted, undirected graph. Algorithms with improved approximation ratios are given for the case when the input is defined on a tree rather than a general graph. The algorithms are also extended to handle time windows whose lengths fall in any bounded range.

Keywords Approximation algorithms · Graph algorithms · Time windows · Repairman problem · Traveling salesman problem

1 Introduction

The traveling salesman problem (TSP) has served as the archetypal hard combinatorial optimization problem that attempts to satisfy requests spread over a metric space [21]. Yet, the TSP is not a perfect model of real life. In particular, a salesman may not
have enough time to visit all desired locations. Furthermore, a visit to any particular location may be of value only if it occurs within a certain specified interval of time. We use the term repairman problem to describe the class of problems that add time constraints to the TSP. We are not the first to pose such problems and note that there has been substantial previous work, which we will survey briefly in the next section.

We consider a fundamental version of such a repairman problem, in which the repairman is presented with a set of service requests. Each service request is located at a node in a weighted, undirected graph and is assigned a time window during which it is valid. Note that multiple service requests may share the same node as a location but have different time windows. The repairman may start at any time from any location and stop similarly. (This latter assumption is at variance with much of the preceding literature about the repairman problem [3, 4] but was made in [2] for the related problem of orienteering (see next section). We choose to frame our problem without specifying initial and final locations because doing so leads to an elegant solution that gives additional insight into such problems. This assumption seems to come at the price of rendering our approach less useful as a general-purpose subroutine.)

We handle two variations of our problem. In the first, when a repairman visits the location of a service request during its time window he performs a service event, and each such event yields a specified profit. A service run is a feasible sequence of service events that a repairman can make at a given speed. The goal of the repairman is to find a service run that satisfies a subset of requests with the maximum total profit possible. On the other hand, a service tour is a service run that satisfies all service requests. Thus, in this second variation, the service provider tries to minimize the speed necessary to make a service tour. Note that there is some minimum speed below which it is not possible to visit all requests. We call this variation the speeding deliveryman problem, recognizing that, for example, a pizza delivery driver may need to hurry to deliver his or her set of orders in a timely manner. We seem to be the first to frame this second problem in terms of speedup, a change from the standard emphasis on distance traveled or profit achieved.

For both variations, we focus primarily on the case in which all time windows are the same length (i.e., unit-time). For the repairman problem, we assume that all profits for service events are identical, although general profits can be accommodated with a small (factor of $1 + o(1)$) increase in the approximation ratio and a polynomial increase in running time, using ideas from [5]. Additionally, we refer to each service event as being instantaneous, although positive service times for a repairman can be absorbed into the structure of the graph. These restrictions still leave problems that are APX-hard for a metric graph, via a simple reduction from TSP, which has been shown to be APX-hard [23].

Our goal is thus to find polynomial-time approximation algorithms. For the repairman, our algorithms produce a service run whose profit is within a constant factor of the profit for an optimal service run. For the deliveryman, they produce a service tour whose maximum speed is within a constant factor of the optimum speed, which accommodates all requests. To the best of our knowledge, we are the first to find approximation algorithms for either problem that get within a constant factor for a general metric, albeit when time windows are the same length. Thus, we establish membership in APX for these specific problem versions.
Our repairman and deliveryman problems are NP-hard even in the case that the service network is an edge-weighted tree rather than a general (weighted) graph, as we shall show. This property is particularly notable, since of course the TSP is polynomial-time solvable on tree networks. In this simpler context of a tree, we give approximation algorithms with improved constants and faster polynomial running times for our problems.

1.1 Related Work

Although we seem to be the first to study the speeding deliveryman problem, we are not the first to consider the repairman problem, which is a generalization of a host of repairman, deliveryman and traveling salesman problems such as those in [3, 4, 8, 19, 24]. Much work has been done on related problems in a metric space on the line. Assuming unit-time windows, a \((4 + \epsilon)\)-approximation was given for the repairman on a line in [4]. We improve this approximation to 3 and in a more general setting, a tree. We are the first to give poly-time constant-ratio algorithms for the unit time window repairman problem on a tree or on a graph.

For general metric spaces and general time windows together in the rooted problem, an \(O(\log^2 n)\)-approximation is given in [3]. An \(O(\log L)\)-approximation is given in [7], for the case that all time window start and end times are integers, where \(L\) is the length of the longest time window. In contrast, a constant approximation is given in [8], but only when there are a constant number of different time windows. Following the initial publication of our work in [13], an extension was given in [7] that gives an \(O(\log D)\)-approximation to the unrooted problem with general time windows, where \(D\) is the ratio of the length of largest time window to the length of the smallest. Poly-logarithmic approximation algorithms to directed TSP with time windows have been given in [9] and [22]. TSP with time windows has also been studied in the operations research community, as in [11] and [12], where it is exhaustively solved to optimality.

The problem of orienteering is also significant because it is used as a subroutine in many deadline and time window problems. In orienteering, the goal is to find a path visiting as many locations as possible, subject to some constraint on the total distance traveled (or time taken). The first significant results in this area found constant approximations for several variations in the plane [1]. A PTAS for orienteering in the plane was later given in [10]. Recent results in rooted and point-to-point versions of orienteering [3, 5, 9] have made the latest improvements in approximation algorithms for time window problems possible.

1.2 Our Results

In this paper, we introduce a time-partitioning scheme that is especially well suited to unit-length time windows. Partitioning was introduced in [3], but that partitioning approach is different because it is designed to handle general time windows in a fashion not intended to get within a constant factor of optimal, even should the windows all be unit-time. In our partitioning, we identify discrete periods of equal length and trim the time window for each request to be the period that was wholly contained
in it. Trimming induces at most a linear number of periods, each of which we can then consider separately. Trimming loses the repairman at most a constant fraction of possible profit and increases the necessary speed of the deliveryman by at most a constant factor. In Sect. 2, we characterize these effects of trimming more thoroughly.

For the variations restricted to trees, once we partition requests on the basis of common periods, we are able, for requests with a common period, to solve a variety of subproblems exactly for the repairman and almost exactly for the deliveryman, in contrast to general graphs for which we use approximate rather than exact solutions. In Sect. 3, we give algorithms that approximate profit for the repairman within a factor of 3 on a tree and within a factor of $6 + \epsilon$ for a graph. For the version on a graph, we use constant approximation algorithms from [3] and [9] as subroutines. We also show ways in which non-zero service times for the repairman can easily be accommodated with small changes to our algorithms. For all of the problems we consider, we can combine solutions for each different period using dynamic programming. Although dynamic programming is used for the deliveryman, the algorithm and especially the analysis differ from the repairman. A key insight is that the effects of trimming can be offset by increasing speed and that the amount of speed needed can be analyzed by imagining the deliveryman running a backwards and forwards pattern along an optimal service tour. In Sect. 4, we give algorithms that approximate speed for the deliveryman within a factor of $4 + \epsilon$ on a tree and within a factor of $\frac{20}{3} + \epsilon$ on a graph.

To deal with windows with lengths between 1 and 2 (or between 1 and some constant $c$), we generalize our repairman algorithms in Sect. 5 by using more than one trimming scheme. Each trimming scheme employs a different period size that, when all such schemes are considered together, adapts to different distributions of window size. By starting each trimming scheme at a number of carefully chosen times and keeping the most profitable run found, we show that the approximation factor for repairman on windows of different lengths can be bounded by a weighted average of the bounds of each trimming scheme. For windows with length between 1 and 2, this bound yields a constant-factor approximation with a better bound than the result in [7] for the same problem.

In the remainder of Sect. 5, we show that a different accounting of trimming allows the speeding deliveryman on windows with length between 1 and 2 (or another bounded length) to be approximated to within a constant factor. Other work [3] has focused on time windows with arbitrary lengths, but improved approximation guarantees for time windows with lengths in some bounded range may be useful for many practical applications in which time window lengths do not vary dramatically.

Finally, in Sect. 6, we sketch the NP-hardness of the problems on a tree. A preliminary version of this paper appeared in [13].

2 Trimming Requests

Trimming is a simple and yet powerful technique that can be applied when we deal with unit-time windows. Starting with time 0, we make divisions in time at values which are integer multiples of one half, i.e., 0, .5, 1, and so on. We assume that no
request window starts on such a division, because if it did, we could redefine times to be decreased by a negligible amount. We thus assume that the starting time for any window is positive. Let a period be the time interval from one division up to but not including the next division. Because every service request is exactly one unit long in time and no request window starts on a division, half of any request window will be wholly contained within only one period, with the rest divided between the preceding and following periods. We then trim each service request window to coincide with the period wholly contained in it, discarding those portions of the request window that fall outside of the chosen period.

For the repairman problem, the trimming may well lower the profit of the best service run, but by no more than a constant factor, as we show below. Let the target interval of a request be that part of the request window that coincides with the period to which the request is trimmed. Call that part of the request window contained in the previous period its early interval, and call that part of the request window contained in the following period its late interval. Let \( \pi(R) \) denote the profit of a service run \( R \).

**Theorem 2.1** (Limited Loss Theorem) Consider any instance of the repairman problem. Let \( R^* \) be an optimal service run with respect to untrimmed requests. There exists a service run \( R \) with respect to trimmed requests such that \( \pi(R) \geq \frac{1}{3} \pi(R^*) \).

**Proof** We use a best-of-three argument. Observe that \( R^* \) must have at least one third of its service events in either the target intervals, the early intervals, or the late intervals. If at least one third of the service events of \( R^* \) occur in target intervals, then have \( R \) follow the same path and schedule as \( R^* \) but service only those requests in target intervals.

If at least one third of the service events of \( R^* \) occur in early intervals, then take service run \( R \) to be \( R^* \) but started .5 units later in time, and with \( R \) servicing those requests that were in early intervals of \( R^* \) but are now in target intervals of \( R \). Then the number of service events of \( R \) will be at least one third of the number of service events for \( R^* \).

Similarly, if at least one third of the service events of \( R^* \) occur in late intervals, take \( R \) to be \( R^* \) but started .5 units earlier in time, with \( R \) servicing those requests that were in late intervals of \( R^* \) but are now in target intervals of \( R \). Recall that, in our version of the problem, run \( R \) may start at any time.

In each case, there is a service run \( R \) for trimmed requests that contains at least one third of the service events of an optimal service run for untrimmed requests. Since one of these three cases must always hold, the desired \( R \) always exists. \( \square \)

For the deliveryman problem, trimming may well increase the necessary speed of the best service tour, but by no more than a constant factor, as we show below. Let \( s(Q) \) denote the minimum speed needed for service tour \( Q \) to visit all service requests.

**Theorem 2.2** (Small Speedup Theorem) Consider any instance of the deliveryman problem. Let \( Q^* \) be an optimal service tour with respect to untrimmed requests start-
ing at time \( t = 0 \). There exists a service tour \( Q \) with respect to trimmed requests such that \( s(Q) \leq 4s(Q^*) \).

**Proof** We shall extend \( Q^* \) backward for \( t < 0 \) by assuming that \( Q^* \) proceeds from any convenient position so that it encounters the original starting position at time \( t = 0 \). Let \textit{racing} describe movement, either forward or backward, along \( Q^* \) at a speed of \( 4s(Q^*) \). We define tour \( Q \) which races along \( Q^* \). During any two consecutive periods, the deliveryman will make a net advance equal to the advance of \( Q^* \) over those two periods.

Identify as \( t_i \) the time \( t = .5i \) which is also the starting time of period \( i \). Let \( f(t) \) be a function that gives the location of the deliveryman on \( Q^* \) for any given time \( t \).

We define \( Q \) as follows. Start tour \( Q \) at \( t = 0 \) at the location that \( Q^* \) has at time \( t = - .5 \). From there, tour \( Q \) follows a repeating pattern of racing forward along \( Q^* \) for 1 period, racing backward along \( Q^* \) for .75 periods, and then racing forward along \( Q^* \) for .25 periods. We define \( q(t) \) to describe the movement of \( Q \) as follows.

For \( ti \leq t < ti + .5 \), where \( i \) is even, define

\[
q(t) = \begin{cases} 
  f(t_i - .5 + 4(t - t_i)) & t_i \leq t \leq t_i + .5 \text{ (forward for 1 period)} \\
  f(t_i + 3.5 - 4(t - t_i)) & t_i + .5 \leq t \leq t_i + .875 \text{ (backward for .75 periods)} \\
  f(t_i - 3.5 + 4(t - t_i)) & t_i + .875 \leq t \leq t_i + 1 \text{ (forward for .25 periods)}
\end{cases}
\]

Figure 1 gives an example of this pattern of movement for some \( Q^* \) and a corresponding \( Q \).

Consider a request \( r \) serviced at time \( t \) in \( Q^* \). If \( ti \leq t < ti + .5 \), then the time window of the request will be trimmed to be one of three periods of length .5: \([ti - .5, ti), [ti, ti + .5), \text{ or } [ti + .5, ti + 1)\). We consider cases when \( i \) is odd or \( i \) is even separately.

**Case 1: \( i \) is odd**

If the window containing \( r \) is trimmed to be \([ti - .5, ti)\), then service the request \( r \) at time \( ti + .25((t - ti) - 1) \). If the window is trimmed to be \([ti, ti + .5)\), then service the request \( r \) at time \( ti + .25((ti - t) + 1) \). If the window is trimmed to be \([ti + .5, ti + 1)\), then service the request \( r \) at time \( ti + .25(((t - ti) + 2) \).

**Case 2: \( i \) is even**

If the window containing \( r \) is trimmed to be \([ti - .5, ti)\), then service the request \( r \) at time \( ti + .25((ti - t) + 1.5) \). If the window is trimmed to be \([ti, ti + .5)\), then service the request \( r \) at time \( ti + .25((ti - t) + 1.5) \). If the window is trimmed to be \([ti + .5, ti + 1)\), then service the request \( r \) at time \( ti + .25((ti - t) + 2) \).
service the request $r$ at time $t_i + .25((t - t_i) + .5)$. If the window is trimmed to be $[t_i + .5, t_i + 1)$, then service the request $r$ at time $t_i + .25((t_i - t) + 3.5)$. □

Although neither the Limited Loss Theorem nor the Small Speedup Theorem give lower bounds on the approximation ratio, we have been unable to identify better bounds, even with the intuition gained from our work in [14].

3 Repairman Problem: Maximizing Profit

3.1 Repairman on a Tree

Trimming is a valuable technique because we can solve the repairman problem on a tree exactly in the case when windows are already trimmed. This exact solution can further be extended to an approximate solution on trimmed windows for the repairman problem on a graph. Both versions of the algorithm will use point-to-point orienteering subroutines within each period to find optimal (or nearly optimal) paths and then use the same dynamic programming approach to paste these partial paths together.

We first give a dynamic programming algorithm for the repairman problem on a tree when all requests share the same time window and have the same profit. To find paths from $s$ to $t$ of all possible profits $p$, we start with the direct path from $s$ to $t$ and then add on low-cost pieces of subtrees that branch off the direct path as necessary to achieve various profits $p$. We do so by contracting the path into a single node $r$ and using dynamic programming to sweep up from the leaves, finding the cheapest paths in the tree for each possible profit.

Our recursive subroutine SWEEP-TREE($r$) produces a list $L_r$ of the lowest costs at which various profit levels can be achieved by including portions of the tree rooted at $r$. List $L_r$ is a mapping from profits to costs where $L_r[p]$ is the cost of achieving profit $p$, if recorded, and $\infty$ otherwise. Let $\pi(u)$ be the profit gained by visiting $u$. Note that $\pi(u)$ counts the number of service requests at $u$. If we define $\pi(r)$ to be the profit of the direct path, then adding $d(s,t)$ to all the costs in the list $L_r$ yields

\[
\text{SWEEP-TREE}(\text{node } u)
\]

For $p$ from 0 to $\pi(u)$, set $L_u[p]$ to be 0.

For each child $v$ of $u$,

Call SWEEP-TREE($v$), which will generate $L_v$.

Add $2d(u,v)$ to each entry in $L_v$ except $L_v[0]$.

Let $\max_u$ be the largest profit in $L_u$ and $\max_v$ the largest profit in $L_v$.

For $p$ from 0 to $\max_u + \max_v$, set $L[p]$ to be $\infty$.

For $a$ from 0 to $\max_u$ and $b$ from 0 to $\max_v$,

Set $L[a + b]$ to be $\min\{L[a + b], L_u[a] + L_v[b]\}$.

Set $L_u$ to be $L$. 
the costs of the best paths on the full tree starting at \( s \) and ending at \( t \) for all possible profit levels.

Let \( n \) be the number of nodes plus the total profit in the tree.

**Lemma 3.1** For all possible profits, SWEEP-TREE identifies minimum-length paths from \( s \) to \( t \) in a total of \( O(n^2) \) time.

Although SWEEP-TREE might be viewed as being reminiscent of Sect. 2.6.3 in [6], we note that the running time claimed there is \( O(nD) \) where \( D \) is a bound on the length of the path found. Since some values of \( D \) will prevent that algorithm from being fully polynomial, we provide SWEEP-TREE for completeness.

Using algorithm SWEEP-TREE, we next give the algorithm REPAIRMAN-TREE for multiple trimmed windows. This algorithm uses dynamic programming to move from period to period, in increasing order by time. As it progresses, it finds service runs of all possible profits from every trimmed request in the current period through some subset of trimmed requests in the current period and arriving at any possible trimmed request in a later period. In this way, for every profit value, we identify the earliest that we can arrive at a request that achieves that profit value. The critical insight is that we may have to leave a certain period rather early in order to reach later requests in time. By recording even such low profit service runs and considering them as starting points, we never rule out a service run that appears to be unpromising in early stages but arrives early enough to visit a large number of requests in later stages.

We focus on those periods that contain at least one trimmed request and number them from \( S_1 \), the period starting at the smallest time value, up to the last period \( S_m \). Let \( R_u^k \) be the earliest arriving \( k \)-profit sequence of service events ending at trimmed request \( u \). Let \( A_u^k \) be the arrival time of \( R_u^k \) at \( u \). For each \( u \), we initialize every \( R_u^1 \) to be \( \{ u \} \) and every \( A_u^1 \) to be 0. For \( k > 1 \), let \( R_u^k \) be initialized to \text{null}, and let every other \( A_u^k \) be initialized to \( \text{begin}(S_{i+1}) \), where \( \text{begin}(S_i) \) is the first time instant in period \( S_i \).

We use SWEEP-TREE to find a path of shortest length from a given starting request to a given ending request, subject to accumulating a specified profit. Let \( \text{time}(R) \) be the amount of time a path \( R \) takes. For each indexed period \( S_i \), from 1 up to \( m \), we process period \( S_i \) as described in PROCESS-PERIOD.

After all the periods have been processed, we identify the largest-profit path found and return that resulting service run \( R \) as the output of algorithm REPAIRMAN-TREE.

**Theorem 3.1** In \( O(n^4) \) time algorithm REPAIRMAN-TREE finds a service run on a tree that has at least \( \frac{1}{3} \) the profit of an optimal service run.

**Proof** Correctness follows because the dynamic programming structure of PROCESS-PERIODS finds a run of optimal profit on trimmed time windows. By the Limited Loss Theorem, trimming time windows reduces the profit found to at worst \( \frac{1}{3} \) of optimal. Note that SWEEP-TREE need be run only once per node per period. Thus, REPAIRMAN-TREE takes \( O(n^4) \) time, making \( O(n^2) \) calls to SWEEP-TREE, each of which takes \( O(n^2) \) time. \( \square \)
PROCESS-PERIOD(period Si)

For each trimmed request u in period Si,

For each possible profit value p,

For each subsequent period Sa that contains a trimmed request,

For each trimmed request v in Sa, do the following:

Let R be the path corresponding to Lr(p) that results from

SWEEP-TREE(r) with s = u and t = v, on the set Si − {u}.

Let $R^−$ be R with its last leg, ending at v, removed.

For k from 1 to $n - \pi(R)$,

If $A^k_u + \text{time}(R^−) < \text{begin}(S_{i+1})$, then

Let profit q be $k + \pi(R) - 1$.

If $A^k_u + \text{time}(R) < A^q_v$, then

Set $R^q_u$ to be $R^k_u$ followed by R.

Set $A^q_v$ to be max{$A^k_u + \text{time}(R), \text{begin}(S_a)$}.

3.2 Repairman on a Graph

We will now extend our approximation algorithm for the repairman problem to a metric graph. We use the improvements to [3] given in [9] and use profit values rather than the distance used in [3, 9]. Our approximation algorithm for the repairman on a graph uses approximations for the following optimization problem which we were able to solve exactly on a tree.

Source-Sink k-Path (k-SSP): Given nodes s and t and integer k, find a path of smallest cost from s to t that contains at least k nodes. (This problem is called min-cost s–t path in [5].)

An algorithm given in [9] which we will call REDUCED-PATH approximates a solution to the k-SSP problem by finding a path from s to t no longer than the optimal path from s to t that collects profit k. The analysis in [9] shows that the output path of REDUCED-PATH collects at least $\frac{k}{\sum + \epsilon}$ profit and runs in $\Lambda(n, \epsilon) = O(n^{O(1/\epsilon^2)})$ time.

Using REDUCED-PATH as a subroutine, we approximate the repairman problem as a whole. Our approximation algorithm for the repairman on a graph incorporates the preceding approximation algorithm within the context of a dynamic programming algorithm with the same overall structure as the algorithm for a tree. For each indexed period $S_i$, from 1 up to $m$, we process period $S_i$ as in PROCESS-PERIOD. The only difference is that instead of taking R to be the path corresponding to $L_r(p)$ that results from SWEEP-TREE(r), we take R to be the output of REDUCED-PATH(u, v, p) on the set $S_i − {u}$, where u is the starting request in the path, v is the ending request in the path, and p is the profit of which the path must have a constant fraction. As before, we identify the largest-profit path found and return the resulting service run R as the output of algorithm REPAIRMAN-GRAPH.
Theorem 3.2 In \(O(n^4 \Lambda (n, \epsilon))\) time, REPAIRMAN-GRAPH finds a service run that has at least \(\frac{1}{6+\epsilon}\) the profit of an optimal service run.

Proof By using the algorithm PROCESS-PERIOD but with the output paths from the REDUCED-PATH algorithm substituted in place of the output of the SWEEP-TREE algorithm, REPAIRMAN-GRAPH will find a most profitable service run formed from the approximately optimal \(k\)-SSP paths. Correctness also follows by the same argument as for Theorem 3.1 with the substitution of approximately optimal paths found inside each period. After taking into account the factor of \(2 + \epsilon\) for the \(k\)-SSP approximation and the factor of 3 for trimming, the service run \(R\) returned by REPAIRMAN-GRAPH has a profit \(\pi(R)\) such that
\[ \pi(R) \geq \frac{1}{6+\epsilon} \pi(R^*). \]

For each period, REPAIRMAN-GRAPH will run REDUCED-PATH \(O(n^3)\) times, to cover all possible starting nodes, ending nodes, and profit values. Since the total number of periods with time windows trimmed into them is no greater than the number of requests, REPAIRMAN-GRAPH will process no more than \(O(n)\) periods, giving a total of \(O(n^4 \Lambda(n, \epsilon))\) time. \(\square\)

Some versions of the traveling repairman problem may require a non-zero service time for each service event. We briefly discuss two natural models for service times. In the first, the interval of service time must be completely contained within the time window for the service event. In the second, the interval of service time needs only to start within the corresponding time window and but not necessarily to finish within the time window.

In the first model, we can easily add a uniform service time \(\mu < 1\) to our solution. For each request, we create a new node connected only to the request node under consideration with an edge requiring \(\frac{\mu}{2}\) time to cross at the given speed. Given that the original request had a time window of \(\left[t, t+1\right]\), we add a request at the new node with a time window of \(\left[t + \frac{\mu}{2}, t + 1 - \frac{\mu}{2}\right]\). Then, we remove the original request. By this construction, we guarantee that there is a delay of \(\mu\) after visiting the original request and that the original request is visited in the correct interval of \(\left[t, t+1-\mu\right]\). Also, after preprocessing, all time windows will be of length \(1-\mu\), satisfying the uniform time window requirement for our algorithms.

In the second model, we can add arbitrary length service times in a similar way. Consider a single service request \(r\) to which we wish to add a service time of length \(\mu_r\). We again add a node connected only to the node in question with an edge requiring \(\frac{\mu_r}{2}\) time to cross at the given speed. Given that the original request has a time window of \(\left[t, t+1\right]\), we add a request to the new node with a time window of \(\left[t + \frac{\mu_r}{2}, t + 1 + \frac{\mu_r}{2}\right]\) and then remove the original request. Again, this construction introduces the necessary delays while maintaining the original constraints and uniform time windows.

4 Deliveryman Problem: Minimizing Speed

4.1 Deliveryman on a Tree

With the Small Speedup Theorem at our disposal, the speeding deliveryman algorithm on a tree is almost as cleanly conceived as the repairman on a tree. When all
requests share the same time window, we can find an optimal solution as follows: Remove any leaf and its adjacent edge if the leaf is not the location of a request, and repeat until every leaf is the location of a request. For a given $u$ and $v$, a solution in this slimmed down tree is to double up every edge that is not the direct path from $u$ to $v$, and then identify the Euler path from $u$ to $v$. The optimal choice of $u$ and $v$ is such that $u$ and $v$ are endpoints of a longest simple path in the slimmed down tree. Such a longest path can be found by a simple dynamic programming algorithm on the slimmed down tree: Root the slimmed down tree at some vertex $r$. Then for each child $w$ of $r$, recursively find the longest path in the subtree rooted at $w$ and also the longest path from any vertex in that subtree to $w$. The time to determine the corresponding information for the subtree rooted at $r$ given the information for the subtrees rooted at its children will be proportional to the number of children. Thus total time is $O(n)$. Given the length of the Euler path, it is easy to compute the minimum necessary speed to visit all requests in the tree during a single period.

To approximate a solution to the problem on a tree over multiple periods, we develop an algorithm to test if a specific speed is fast enough to visit all requests during their periods. We use the idea behind the single-period solution in conjunction with dynamic programming. TEST-SPEED processes every period $S_i$ in order and finds, for every pair of requests $u$ and $v$, the earliest-arriving paths that start at request $u$, end at request $v$, and visit all requests in the period. It then glues each of these paths to the earliest-arriving paths which visit all requests in previous periods and, for every request $v$ in $S_i$, keeps the earliest-arriving complete path ending at $v$. Let $\epsilon$ be an input parameter and $\epsilon' = \epsilon/4$. Define the algorithm DELIVERY-TREE, which binary searches $\log \frac{1}{\epsilon'}$ times on a range of speeds using TEST-SPEED, and then, from among the paths found by TEST-SPEED that visit all requests, returns the path $Q$ that uses the slowest speed.

For $u \in S_i$, let the arrival time $A_u$ be the earliest time at which a path visiting all the requests in periods before $S_i$ arrives at request $u$ before visiting any other request in $S_i$. For $v \in S_i$, let the departure time $D_v$ be the earliest time at which a path visiting all requests in the periods up to and including $S_i$ ends at request $v$. For $u, v \in S_i$, let $\text{length}_{uv}$ be the length of the shortest path from $u$ to $v$ which visits all requests in $S_i$.

**Theorem 4.1** For any $\epsilon > 0$, in $O(n^2 \log \frac{1}{\epsilon})$ time DELIVERY-TREE finds a service tour of speed at most $4 + \epsilon$ times the optimal speed.

**Proof** Correctness follows because an earliest-arriving path found by TEST-SPEED that visits all requests during their periods implies that the speed is sufficiently fast. No optimal service tour $Q^*$ can be shorter than the sum of the weights of the subtrees induced by each period plus the shortest distances between each consecutive subtree. We can find this length in $O(n^2)$ time and make a lower bound for $s(Q^*)$. Likewise, $Q^*$ cannot be longer than four times the sum of the weights of the subtrees induced by each period plus the shortest distances between each consecutive subtree. (Use the first length of each subtree to get to any vertex $u$, use double the length of the subtree to traverse it and end at some vertex $v$, and use the last length of the tree to get to the terminus of the shortest path to the next subtree.) We can also find this length in $O(n^2)$ time and make an upper bound for $s(Q^*)$. Using TEST-SPEED, our
TEST-SPEED(speed)

For each request $u$ in $S_1$, set $A_u$ to be 0.

For $i$ from 1 to $m$,

For each request $v$ in $S_i$,

Set $D_v$ to be $\min_{u \in S_i} \{A_u + \text{length}_{uv}/\text{speed}\}$.

If $D_v >$ last time instant of $S_i$, then set $D_v$ to be $\infty$.

For each request $w$ in $S_{i+1}$,

Set $A_w$ to be $\max\{\text{first time instant of } S_{i+1}, \min_{v \in S_i} \{D_v + d(v, w)/\text{speed}\}\}$.

If $A_w >$ last time instant of $S_{i+1}$, then set $A_w$ to be $\infty$.

If there exists a request $v \in S_m$ such that $D_v < \infty$,

then return “feasible speed”, else return “speed too slow”.

algorithm DELIVERY-TREE binary searches in this range to find a speed within a factor of $1 + \epsilon'$ of the optimal speed. Since TEST-SPEED takes $O(n^2)$ to run, we get a total of $O(n^2 \log \frac{1}{\epsilon'})$ time for DELIVERY-TREE. Using the Small Speedup Theorem applies a factor of 4, yielding a $(4 + \epsilon)$-approximation. $\square$

4.2 Deliveryman on a Graph

The algorithm for the deliveryman on a graph is essentially the same as for a tree except that in TEST-SPEED we replace length$_{uv}$, the length of a shortest path in a subtree starting at $u$ and ending at $v$ and visiting every each request vertex, with the length of an approximately optimal path in a subgraph starting at $u$ and ending at $v$ and visiting every each request vertex. We can find this approximately optimal path from $u$ to $v$ using an adaptation of Christofides heuristic to paths, given in [17], which achieves an approximation ratio of $5/3$.

Define the algorithm DELIVERY-GRAPH, which for each period $S_i$ finds an approximately optimal path from $u$ to $v$ for all pairs of request vertices $u$ and $v$ in period $S_i$. Similar to DELIVERY-TREE, we can use a minimum spanning tree for each subgraph to establish upper and lower bounds on the speed of the optimal tour $Q^*$. Then, DELIVERY-GRAPH binary searches $\log \frac{1}{\epsilon'}$ times on a range of speeds using TEST-SPEED modified to use the length of the approximately optimal path instead of length$_{uv}$, and then, from among the paths found by TEST-SPEED that visit all requests, returns the path $Q$ that uses the slowest speed.

Let $Q^*_i$ be the subtour of tour $Q^*$ restricted to requests inside period $S_i$, and $Q_i$ be the subtour of tour $Q$ restricted to $S_i$. Let $u^*_i$ be the first node in subtour $Q^*_i$ and $v^*_i$ the last. Similarly, let $u_i$ be the first node in subtour $Q_i$ and $v_i$ the last.

Lemma 4.1 Let $s$ be the minimum required speed for $Q^*$. Suppose a deliveryman $M^*$ is traveling along $Q^*$ at speed $\eta$ while another deliveryman $M$ is traveling along $Q$ at a speed that never exceeds $\frac{5}{3}\eta$. Then for each $S_i$ deliveryman $M^*$ will never arrive at the first node in $Q^*_i$ before deliveryman $M$ arrives at the first node in $Q_i$. $\square$
Proof By induction on $i$.

Basis: ($i = 1$)
Deliverymen $M$ and $M^*$ start tours $Q$ and $Q^*$, respectively, at the same time.

Induction Step: ($i > 1$)
The length of subtour $Q^*_{i-1}$ is at least $3/5$ the length of subtour $Q_{i-1}$. The distance from $v^*_{i-1}$ to $u^*_i$ is never shorter than the distance from $v_{i-1}$ to $u_i$. Thus, the total distance from $u^*_{i-1}$ to $u_i$ in $Q^*$ is never less than $3/5$ the total distance from $u_{i-1}$ to $u_i$ in $Q$. Since deliveryman $M$ is traveling $Q$ at a maximum speed which is $5/3$ the maximum speed that deliveryman $M^*$ travels $Q^*$, $M$ arrives at $u_i$ no later than $D^*$ arrives at $u^*_i$.

\[ \square \]

Theorem 4.2 In $O(n^5 + n^2 \log \frac{1}{\epsilon})$ time DELIVERY-GRAPH finds a service tour of speed at most $\frac{20}{3} + \epsilon$ times the optimal speed.

Proof Correctness for DELIVERY-GRAPH is the same as DELIVERY-TREE. Running the $O(n^3)$ path heuristic from [17] for $O(n^2)$ pairs gives a total of $O(n^5)$ to find the paths on subgraphs in each period. Since TEST-SPEED takes $O(n^2)$ for each iteration of binary search, we get a total of $O(n^5 + n^2 \log \frac{1}{\epsilon})$ time for DELIVERY-GRAPH. The factor of $5/3$ given by Lemma 4.1 multiplied by the factor of $4$ given by the Small Speedup Theorem for trimming shows that DELIVERY-GRAPH returns a service tour $Q$ such that $s(Q) \leq (\frac{20}{3} + \epsilon) s(Q^*)$, where $Q^*$ is an optimal service tour.

\[ \square \]

5 Problems with Windows of Different Lengths

In this section we consider our approximation problems for windows with length between 1 and 2 and then explain how the same ideas can be extended to general time windows.

5.1 Repairman with Windows of Different Lengths

In this subsection we present an algorithm that achieves a constant-factor approximation for the traveling repairman on windows with length between 1 and 2 and then explain how the same ideas can be extended to general time windows. An $O(\log^2 n)$-approximation is given in [3] for the general, rooted repairman problem. In the case of integral edge lengths and window release and deadline times, an $O(\log D_{\text{max}})$-approximation, where $D_{\text{max}}$ is the latest time a time window ends, is also given in [3] for the general, rooted repairman problem. After the initial publication of our work on unit time windows in [13], an extension to windows with length between 1 and 2 was given in [7] for the unrooted repairman problem. Under the same assumptions, the better bound of $O(\log L)$ is achieved in [7], where all time windows have integral length at most $L$.

Here we present improved approximation factors for the case that windows have length between 1 and 2. It was also claimed in [7] that a constant approximation for this case allows a $O(\log D)$-approximation for the general, unrooted repairman.
problem, where \( D \) is the ratio of the length of the longest time window to the length of the shortest time window. We give an algorithm and analysis for the general, unrooted problem with improved constants, though still a \( O(\log D) \)-approximation.

Below we describe the algorithm WINDOW12 which allows us to approximate the repairman problem when windows have length between 1 and 2. To unify notation, let \( \Gamma(n) \) represent the running time for repairman approximations with trimmed windows on either a metric graph or a tree, as appropriate. As shown in Theorems 3.1 and 3.2, \( \Gamma(n) = O(n^4) \) for a tree and \( O(n^4 \Lambda(n, \epsilon)) \) for a metric graph. Likewise, let the approximation ratios for the repairman algorithms used after trimming be \( \gamma \), where \( \gamma = 1 \) for a tree, where \( k \)-SSP can be solved optimally, and \( \gamma = 2 + \epsilon \) for a metric graph, as shown in [9].

As pointed out in [7], a relatively simple extension of our Limited Loss Theorem allows one to achieve a \( 5\gamma \)-approximation when windows have length between 1 and 2. A more refined approach that we now describe will improve substantially on this constant. Our approach is to try several different sizes for periods. When most of the windows are of length closer to 1, then a period size of 1/2 works well. When most of the windows are of length closer to 2, then a period size of 1 works well. When many of the windows are of length closer to 3/2, then a period size of 3/4 works well.

For each period size, we will consider multiple starting points for a set of periods, each spaced 1/4 apart. Thus, sets of periods whose period sizes are 1/2, 3/4, and 1 will have 2, 3, and 4 unique starting positions, respectively. Depending on a given period size and starting point, a window will partially fill 2 subintervals and fully fill 0, 1, 2, or 3 subintervals between the 2 partial intervals. Let \( W_\ell \) be the set of windows that completely fills exactly \( \ell \) subintervals and partially overlaps with two more of them.

When trimming, we may have to select from among several choices of which single full subinterval to keep for each window. For example, for periods of length 1/2 and for windows in \( W_3 \) which would have three full subintervals, the choices for trimming will be the first, second, or third full subinterval. Combining these choices with the two choices associated with windows in \( W_2 \) and the single choice in windows in \( W_1 \) would yield 6 trimmings, or, in general, \( k! \) where \( k \) is the largest number of subintervals completely filled. Let REPAIR be the appropriate basic repairman algorithm on trimmed windows, either for a tree or for a metric graph, described in Sect. 3. For each period size, for each starting point, for each choice of trimming, we will run REPAIR and keep the result if the profit is better than a previous run.

When window lengths are not all the same, our analysis depends on an averaging argument. By using many service runs based on an optimal run, we can record the total number of times a given interval is visited by all runs. From all the intervals of all the windows, we find one that is visited the least. The number of times this interval is visited divided by the total number of runs is a lower bound on the fraction of profit collected, relative to optimal.

We analyze the performance of WINDOW12 as follows. Let \( R^* \) be an optimal service run for a repairman instance with time window lengths from 1 up to but not including 2. For the sake of analysis, we introduce a new set of periods with duration 1/4. If we split each window into subintervals of length 1/4 along boundaries of
**PHASE 1:**
Set the period size to 1/2 and identify windows for sets $W_1$, $W_2$, and $W_3$.
For $i$ from 0 to 1,
Set the starting point for the periods to $i/4$.
For $j$ from 1 to 2,
For $k$ from 1 to 3,
Trim each window in $W_1$ to its 1st full subinterval.
Trim each window in $W_2$ to its $j$th full subinterval.
Trim each window in $W_3$ to its $k$th full subinterval.
Run REPAIR and retain the best result so far.

**PHASE 2:**
Set the period size to 3/4 and identify windows for $W_1$ and $W_2$.
For $i$ from 0 to 2,
Set the starting point for the periods to $i/4$.
For $j$ from 1 to 2,
Trim each window in $W_1$ to its 1st full subinterval.
Trim each window in $W_2$ to its $j$th full subinterval.
Run REPAIR and retain the best result so far.

**PHASE 3:**
Set the period size to 1 and identify windows for $W_1$.
For $i$ from 0 to 3,
Set the starting point for the periods to $i/4$.
Trim each window in $W_1$ to its 1st full subinterval.
Run REPAIR and retain the best result so far.

These new periods, we get windows in sets $H_3$, $H_4$, $H_5$, $H_6$, and $H_7$. Let the total fraction of profit in an optimal solution coming from windows in set $H_\ell$ be $h_\ell$. Thus, $\sum_{\ell=3}^{7} h_\ell = 1$.

We use these subintervals to give a finer granularity when analyzing the performance of the algorithm run on periods of greater length, viz. 1/2, 3/4, and 1. Consider set $H_\ell$ of windows, $\ell = 3, 4, 5, 6, 7$ and period length $j/4$, for $j = 1, 2, 3, 4$.

Then, the number of full subintervals of a window in $H_\ell$ that are covered when the period length is $j/4$ is either $\lfloor (\ell - j - 1)/j \rfloor$ or $\lceil (\ell - j - 1)/j \rceil$ depending on which set of periods is used. The average coverage over all sets of periods with period length $j/4$ is $(\ell - j - 1)/j$. As before, let $\gamma$ be the approximation bound on the basic repairman algorithm on unit-time windows when trimming has already been done.

**Lemma 5.1** Let $w_\ell$ be the fraction of total profit gained by visiting windows of type $W_\ell$ with an optimal path on untrimmed windows. Let $W$ be a collection of sets $W_\ell$
given by trimming windows in an instance of the repairman problem. The fraction of profit for the best run on trimmed windows in that instance is at least

\[ \frac{1}{\gamma} \sum_{\ell \in \mathcal{W}} \frac{w_\ell}{\ell + 2} \]

**Proof** Each window in set \( W_\ell \) is divided into \( \ell + 2 \) different subintervals. Label the fraction of profit from each of these respective subintervals \( w^{(i)}_\ell \) through \( w^{(\ell + 2)}_\ell \). Note that \( \sum_{i=1}^{\ell+2} w^{(i)}_\ell = w_\ell \).

Let \( W_k \) be the set of windows in \( \mathcal{W} \) which can be divided into the largest number of subintervals. For \( W_k \), there is a subinterval \( i \) such that \( w^{(i)}_k \leq w_k/(k + 2) \). Ignore that subinterval and pair up the \( k + 1 \) subintervals from \( W_k - 1 \) in increasing order of index with the \( k + 1 \) remaining subintervals from \( W_k \). The total profit from these \( k + 1 \) pairs is at least \( \pi = w_k - 1 + (k + 1)w_k/(k + 2) \). Of these pairs, there is a pair whose profit is no greater than \( \pi/(k + 1) \). Then, ignore this pair and, in increasing order of index, match up the \( k \) subintervals from \( W_k - 2 \) with the \( k \) remaining pairs. Repeat this process of ignoring a smallest tuple for a given \( \ell \) and matching it up with the \( \ell + 1 \) subintervals from the next set of smaller windows \( W_{\ell - 1} \).

At the end of the process, the remaining \( k \)-tuple will have a fraction of profit that is

\[ \sum_{\ell=1}^{k} \frac{(\ell + 1)!w_\ell}{(\ell + 2)!} = \sum_{\ell \in \mathcal{W}} \frac{w_\ell}{\ell + 2} \]  

\( \square \)

**Lemma 5.2** From among the two sets of periods and among the six different trimmings created in the first phase of WINDOW12, one such pair of choices yields a run \( R \) such that \( \pi(R)/\pi(R^*) \geq (h_3/3 + 7h_4/24 + h_5/4 + 9h_6/40 + h_7/5)/\gamma \).

**Proof** Consider the trimmed windows from two different shifts of periods from the first phase of WINDOW12. By application of Lemma 5.1, we find the following contributions. Windows from \( H_3 \) will contribute \( h_3/3 \) in both sets of periods. Windows from \( H_4 \) will contribute \( h_4/3 \) in one set of periods and \( h_4/4 \) in the other. Windows from \( H_5 \) will contribute \( h_5/4 \) in both sets of periods. Windows from \( H_6 \) will contribute \( h_6/4 \) in one set of periods and \( h_6/5 \) in the other. Finally, windows from set \( H_7 \) will contribute \( h_7/5 \) in both sets of periods. When the values for both sets of periods are averaged together, the final result satisfies \( \pi(R)/\pi(R^*) \geq (h_3/3 + 7h_4/24 + h_5/4 + h_6/8 + h_6/10 + h_7/5)/\gamma = (h_3/3 + 7h_4/24 + h_5/4 + 9h_6/40 + h_7/5)/\gamma \).  

\( \square \)

**Lemma 5.3** From among the three sets of periods and among the two different trimmings created in the second phase of WINDOW12, one such pair of choices yields a run \( R \) such that \( \pi(R)/\pi(R^*) \geq (h_3/9 + 2h_4/9 + h_5/3 + 11h_6/36 + 5h_7/18)/\gamma \).

**Proof** Consider the trimmed windows from three different shifts of periods from the second phase of WINDOW12. By application of Lemma 5.1, we find the following...
contributions. Windows from \(H_3\) will contribute \(h_3/3\) in one set of periods and nothing in the other two. Windows from \(H_4\) will contribute \(h_4/3\) in two sets of periods and nothing in the other one. Window from \(H_5\) will contribute \(h_5/3\) in all three sets of periods. Windows from \(H_6\) will contribute \(h_6/3\) in two sets of periods and \(h_6/4\) in the other one. Windows from \(H_7\) will contribute \(h_7/3\) in one set of periods and \(h_7/4\) in the other two. Averaging the values for all three sets of periods gives the claimed result.

\[\square\]

**Lemma 5.4** From among the four sets of periods created in the third phase of \(\text{WINDOW12}\), one of them yields a run \(R\) such that \(\pi(R)/\pi(R^*) \geq (h_4/12 + h_5/6 + h_6/4 + h_7/3)/\gamma\).

**Proof** Consider the trimmed windows from four different shifts of periods from the third phase of \(\text{WINDOW12}\). By application of Lemma 5.1, we find the following contributions. Windows from \(H_3\) will contribute nothing in all four sets of periods. Windows from \(H_4\) will contribute \(h_4/3\) in one set of periods and nothing in the other three. Windows from \(H_5\) will contribute \(h_5/3\) in two sets of periods and nothing in the other two. Windows from \(H_6\) will contribute \(h_6/3\) in three sets of periods and nothing in the other one. Windows from \(H_7\) will contribute \(h_7/3\) in all four sets of periods. Averaging the values for all four sets of periods gives the claimed result.

\[\square\]

**Theorem 5.1** In \(O(\Gamma(n))\) time, algorithm \(\text{WINDOW12}\) identifies a run \(R\) such that \(\pi(R)/\pi(R^*) \geq 52/(219\gamma)\).

**Proof** By Lemmas 5.2, 5.3, and 5.4, \(\pi(R)/\pi(R^*) \geq \frac{1}{\gamma} \max \left\{ \frac{x(h_3/3 + 7h_4/24 + h_5/4 + 9h_6/40 + h_7/5, h_3/9 + 2h_4/9 + h_5/3 + 11h_6/36 + 5h_7/18, h_4/12 + h_5/6 + h_6/4 + h_7/3)}{x + y + z = 1} \right\} \)

This expression achieves a maximum for \(x = 50/73, y = 6/73, \) and \(z = 17/73\), yielding

\[\pi(R)/\pi(R^*) \geq \frac{52(h_3 + h_4 + h_5 + h_6 + h_7)}{219\gamma} = \frac{52}{219\gamma}\]

Algorithm \(\text{WINDOW12}\) runs the basic repairman algorithm 12 times in the first phase, 6 times in the second phase, and 4 times in the third phase, for a total of 22 times. Since the running time of the basic repairman algorithm is \(\Gamma(n)\), the running time of \(\text{WINDOW12}\) is \(O(\Gamma(n))\).

\[\square\]

For handling requests whose windows are between 1 and 2, the best performance ratio that we have achieved is \(219\gamma/52\). To handle windows whose largest length \(D\) is either greater than or less than \(2\), we have identified two additional techniques.
The first technique, applicable for \( D > 2 \), partitions the windows into sets such that the lengths within each set will be within a factor of 2 of each other. We then run WINDOW12 on each such set and choose the best result. This idea of partitioning windows into sets corresponding to lengths in ranges bounded by consecutive powers of 2 was also used in [4].

The second technique, applicable to either \( D < 2 \) or \( D > 2 \), extends the approach of WINDOW12. For \( D \) sufficiently smaller than 2, we will use a proportionately smaller distance between the beginnings of periods, and, rather than use periods of length \( 1/2, 3/4, \) and 1, use periods of length \( 1/2, 1/2 + 1/2^k, \) and \( 1/2 + 1/2^k \). For \( D \) sufficiently larger than 2, we will use more than three different period lengths. For \( D = 3 \), for example, we would use period lengths of \( 1/2, 3/4, 1, 5/4, \) and \( 3/2 \).

As the number of period lengths increases, the time to consider various combinations grows exponentially. It thus makes sense, when \( D > 2 \), to use a combination of the first and second techniques. We will analyze the performance with this in mind.

We now present a generalized version of WINDOW12 called WINDOWG which can be applied to windows of any length between 1 and \( 1 + p/2^g \) for natural numbers \( p \) and \( g \). Let \( q = 1/2^{g+1} \). Let \( P_0 \) be a set of periods of length \( q \).

For \( i = 0, 1, \ldots, p \), let \( P_i^{(0)}, P_i^{(1)}, \ldots, P_i^{(i)} \) be sets of periods of length \( (i + 2^g)q \), where the first period of \( P_i^{(0)} \) begins at the same instant as the first period of \( P_0 \), and for each \( j = 1, 2, \ldots, i \), the first period of \( P_i^{(j)} \) begins \( q \) after the first period of \( P_i^{(j-1)} \).

It is convenient to use the factorial number system [20, p. 175], which we review here. In this system, a nonnegative integer is represented by a sequence of digits \( d_u \ldots d_2 d_1 \) where \( d_i \in \{0, 1, \ldots, i\} \). The value of \( v \) is \( \sum_{i=1}^{u} i! \cdot d_i \). Every value is uniquely represented, and it follows that \( 10\ldots0 \) is \( u! \).

**Lemma 5.5** For integers \( g \geq 1 \) and \( p \geq 1 \), the running time of WINDOWG is \( O((p + g)! \cdot \Gamma(n)) \).

**Proof** From inspection, the running time of WINDOWG is proportional to at most

\[
\sum_{i=0}^{p} \frac{(p+g-i)!}{(p+g)!} \sum_{k=0}^{(p+g-i)-1} \Gamma(n) = \Gamma(n)(p+g)! \sum_{i=0}^{p} (i+2) \frac{(p+g-i)!}{(p+g)!}
\]

We bound the value of the summation by a constant. Since \( p+g-(i-1) \geq g+1 \geq 2 \) for any \( i \leq p \),

\[
\sum_{i=0}^{p} (i+2) \frac{(p+g-i)!}{(p+g)!} \leq \sum_{i=0}^{p} \frac{(i+2)}{2^i} < \sum_{i=0}^{\infty} \frac{2^i}{2^i} = 6 \quad \square
\]

Note that when \( p \geq 2 \), there appears to be no benefit to having a value of \( g > 1 \) except where needed to specify a precise fractional value for maximum window length. We do not suggest allowing \( p \) or \( g \) to range freely, since factorial growth is unacceptable. However, we can fix values of \( p \) and \( g \) to give an appropriate running
For $i$ from 0 to $p$ (consider periods of length $(i + 2^g)q$),
For $j$ from 0 to $i + 1$ (consider sets $P_i^{(j)}$ of periods),
For $k$ from 0 to $(p + g - i)! - 1$ (choose trim positions),
Let $d_u \cdots d_1$ be the representation of $k$ in the factorial number system.
Assume $d_0 = 0$.
For each request $x$,
Let $v$ be the number of periods of length $(i + 2^g)q$ that $x$ has in $P_i^{(j)}$.
If $v > 0$ then
Let $w = 1 + d_{v-1}$.
Trim the window of $x$ to its $w$th period of length $(i + 2^g)q$ in $P_i^{(j)}$.
Else
Exclude $x$ from this run.
Run REPAIR and retain the best result so far.

Time and then partition the windows into sets such that the first partition contains windows whose lengths are between 1 and $1 + p/2^g$, the second set contains windows whose lengths are between $1 + p/2^g$ and $(1 + p/2^g)^2$, the third between $(1 + p/2^g)^2$ and $(1 + p/2^g)^3$, and so on. Let $D$ be the ratio of the longest window length to the shortest. Let $b = 1 + p/2^g$. Then, there are $\log_b D$ such sets. Let WINDOWGD be the algorithm that runs WINDOWG on each of the $\log_b D$ sets separately and returns the highest profit run found.

**Theorem 5.2** Let $p$ and $g$ be fixed and $b = 1 + p/2^g$. Let $D$ be the ratio of the longest window length to the shortest. Let the approximation ratio of WINDOWG for windows between 1 and $b$ be a function of $p$ and $g$ given by $\rho(p, g)\gamma$. The approximation ratio of WINDOWGD for windows of general length is $\rho(p, g)\gamma \log_b D$.

**Proof** Since the union of all of the $\log_b D$ disjoint sets is the set of all windows, one set must contain at least $1/\log_b D$ of the requests serviced by an optimal run. Since the approximation ratio for each set is $\rho(p, g)\gamma$, the profit found on that set is within a factor of $\rho(p, g)\gamma \log_b D$ of optimal. \qed

Values of $\rho(p, g)$ can be calculated for inputs $p$ and $g$ using a linear program. For example, for maximum window sizes of 2, 3, and 4, the values of $\rho(p, g)$ are $219/32 \approx 4.2115$, $24619/4954 \approx 4.9695$, and $1427019/258044 \approx 5.5301$, respectively. When $b = 2$, our result is a $4.2115\gamma \log D$ approximation, an improvement over the $5\gamma \log D$ approximation given in [7].

### 5.2 Deliveryman with Windows of Different Lengths

In this subsection we expand our analysis for the deliveryman problem to windows with length between 1 and 2, and also to windows with length in any bounded range.
To unify notation, let $\Delta(n)$ represent the running time for deliveryman approximations using trimmed windows either on a metric graph or on a tree, as appropriate. As shown in Theorems 4.1 and 4.2, $\Delta(n)$ is $O(n^2 \log \frac{1}{\epsilon})$ for a tree and $O(n^5 + n^2 \log \frac{1}{\epsilon})$ for a metric graph. Likewise, let the approximation ratios for the algorithms used after trimming be $\delta$, where $\delta = 1 + \epsilon$ for a tree and $\delta = 5/3 + \epsilon$ for a metric graph, as shown by the same theorems.

We now consider the version of the deliveryman problem which allows the lengths of request time windows to range over the interval $[1, 2)$ instead of being confined to unit size. We trim the time window to the first period wholly contained within it. Of course, trimming may increase the necessary speed of the best service tour, but by no more than a constant factor.

**Lemma 5.6** Let $Q^*$ be an optimal service tour with respect to untrimmed requests whose lengths are in the interval $[1, 2)$. There exists a service tour $Q$ with respect to trimmed requests such that $s(Q) \leq 6s(Q^*)$.

**Proof** The proof is similar to that of Theorem 2.2, with minor changes to ensure that $Q^*$ hits all five, instead of what was previously three, intervals that correspond to the at most five .5 length periods with which a time window could intersect.

We shall extend $Q^*$ backward for $t < 0$ by assuming that $Q^*$ proceeds from any convenient position so that it encounters the original starting position at time $t = 0$. Let racing now be movement, either forward or backward, along $Q^*$ at a speed of $6s(Q^*)$ instead of $4s(Q^*)$. We define tour $Q$ which races along $Q^*$. During any two consecutive periods, the deliveryman will make a net advance equal to the advance of $Q^*$ over those two periods.

Identify as $t_i$ the time $t = .5i$ which is also the starting time of period $i$. As in the proof of Theorem 2.2, we shall extend $Q$ backward for $t < 0$ by assuming that $Q$ proceeds from any convenient position so that it encounters the original starting position at time $t = 0$. We define $Q$ as follows. Start tour $Q$ at $t = 0$ at the location that $Q^*$ has at time $t = -1.5$. From there, tour $Q$ follows a repeating pattern of racing forward along $Q^*$ for 1 period, racing backward along $Q^*$ for $\frac{5}{6}$ periods, and racing forward again along $Q^*$ for a final $\frac{1}{6}$ period.

As in the proof of Theorem 2.2, we consider a request $r$ serviced at time $t$ by $Q^*$. If $t_i \leq t < t_i + .5$, then the time window of the request will be trimmed to be one of five periods of length $.5$: $[t_i - .5, t_i)$, $[t_i, t_i + .5)$, $[t_i + .5, t_i + 1)$, $[t_i + 1, t_i + 1.5)$, and $[t_i + 1.5, t_i + 2)$. We consider cases when $i$ is odd or $i$ is even separately.

**Case 1:** $i$ is odd

If the window containing $r$ is trimmed to be $[t_i - .5, t_i)$, then service the request $r$ at time $t_i + (1/6)((t - t_i) - 1)$. If the window is trimmed to be $[t_i, t_i + .5)$, then service the request $r$ at time $t_i + (1/6)((t_i - t) + 1)$. If the window is trimmed to be $[t_i + .5, t_i + 1)$, then service the request $r$ at time $t_i + (1/6)((t_i - t) + 4)$. If the window is trimmed to be $[t_i + 1, t_i + 1.5)$, then service the request $r$ at time $t_i + (1/6)((t_i - t) + 8)$. If the window is trimmed to be $[t_i + 1.5, t_i + 2)$, then service the request $r$ at time $t_i + (1/6)((t_i - t) + 9).

**Case 2:** $i$ is even
If the window containing \( r \) is trimmed to be \([t_i - .5, t_i]\), then service the request \( r \) at time \( t_i + (1/6)((t_i - t) - 5/2) \). If the window is trimmed to be \([t_i, t_i + .5]\), then service the request \( r \) at time \( t_i + (1/6)((t - t_i) + 3/2) \). If the window is trimmed to be \([t_i + .5, t_i + 1]\), then service the request \( r \) at time \( t_i + (1/6)((t_i - t) + 9/2) \).

If the window is trimmed to be \([t_i + 1, t_i + 1.5]\), then service the request \( r \) at time \( t_i + (1/6)((t - t_i) + 13/2) \). If the window is trimmed to be \([t_i + 1.5, t_i + 2]\), then service the request \( r \) at time \( t_i + (1/6)((t_i - t) + 23/2) \).

Let DELIVERY be the appropriate deliveryman algorithm on trimmed windows, either for a tree or for a metric graph, described in Sect. 4.

**Theorem 5.3** In \( O(\Delta(n)) \) time DELIVERY finds a service tour of speed at most \( 6\delta \) times the optimal speed for windows with length between 1 and 2.

**Proof** Correctness follows from Theorems 4.1 and 4.2. The factor of 6 for trimming given by Lemma 5.6 multiplied by the appropriate approximation factor \( \delta \) after trimming shows that DELIVERY returns a service tour \( Q \) such that \( s(Q) \leq 6\delta s(Q^*) \), where \( Q^* \) is an optimal service tour. Since a deliveryman algorithm on trimmed windows is only run a single time, the running time is \( O(\Delta(n)) \).

Observe that this pattern can be extrapolated to windows with arbitrary lengths. As before, let \( D \) be the ratio of the longest time window to the shortest. In the event that the ratio is not an integer, let \( D \) be the ceiling of the ratio. Let the speed be \( 2D + 2 \). Define \( Q \) as a generalization of the previous definition. Start tour \( Q \) at \( t = 0 \) at the location that \( Q^* \) has at time \( t = .5 - D \). From there, tour \( Q \) follows a repeating pattern of racing forward along \( Q^* \) for 1 period, racing backward along \( Q^* \) for \( \frac{2D+1}{2D+2} \) periods, and racing forward again along \( Q^* \) for a final \( \frac{1}{2D+2} \) periods. Clearly, \( Q \) will hit all \( 2D + 1 \) intervals of length .5 that a window can intersect with while making the required progress of \((2D + 2)(1 - \frac{2D+1}{2D+2} + \frac{1}{2D+2}) = 2 \) periods.

### 6 NP-hardness on a Tree

By a reduction to TSP, the traveling repairman problem with unit-time windows is APX-hard on a weighted, metric graph. For the case of a line, NP-completeness proofs for many of the time-constrained traveling salesman problems were given in [24], but we know of no proof that the unit-time window repairman problem on a line given in [4] is NP-hard. Below, we consider the hardness of repairman when the problem is on a tree.

**Theorem 6.1** The traveling repairman problem with equal-length time windows is NP-hard on tree metrics.

**Proof** We use a reduction from a version of the partition problem restricted to positive values: Given a multiset of \( n \) positive integers, decide whether the multiset can be
partitioned into two multisets which sum to the same value, i.e. half the sum of all of the integers in the multiset. This problem is NP-complete [18].

Our reduction is as follows. We assume that the sum of the integers in the multiset is $2K$. First create a central node $u$ in a tree by itself. For each integer in the multiset, create a node and connect it to $u$ with an edge having the cost of the integer. Then create the start and end nodes $s$ and $t$ at the same location and connect them to $u$ with an edge of cost $6K$. Also create the midpoint node $v$ and connect it to $u$ with an edge of cost $K$. To complete the input for the repairman problem, we must also create service requests having both a time window and a node for a location. Create a service request with time window $[0, 6K]$ located at $s$ and a request with window $[12K, 18K]$ located at $t$. Create requests located at each of the nodes corresponding to an integer and also at the central node $u$ all with time window $[6K, 12K]$. Finally, create two requests located at the midpoint node $v$, one with time window $[3K, 9K]$ and the other with time window $[9K, 15K]$. Recall that our definition of service requests allows multiple requests to share a single node as a location. Note that the graph is a tree and that all the time windows are exactly $6K$ units long, meeting the unit-length requirement. The optimal tour has just enough time to visit all of the nodes if and only if it starts at the start node $s$ at time 0, reaches node $u$ at time $6K$, visits a set of nodes whose integers sum to $K$, returning to node $u$ by time $8K$, visits the midpoint node $v$ at time $9K$, returning to node $u$ by time $10K$, visits the remaining nodes which also sum to exactly $K$, returning to node $u$ by time $12K$, and finally ends at the end node $t$ at time $18K$.

Figure 2 gives an illustration of the reduction for the set $\{2, 3, 4, 5, 6\}$. The tree node for each value in the set is named with that value, and has its time window labeled above. Each edge from node $u$ to a node not labeled with a value is labeled with its length. In this example, $K = 10$. The repairman leaves node $s$ at time 0, visits node $u$ for the first time at time 60, visits node $v$ at time 90 (during both associated time windows), visits node $u$ for the last time at time 120, and visits node $t$ at time 180. The set has a partition into $\{2, 3, 5\}$ and $\{4, 6\}$ and the set of requests associated with each of these subsets can be accommodated either before or after the visit to node $v$.

![Figure 2](image_url)
While this proof assumes intervals closed on both ends, it is trivial to modify the proof for intervals closed on one end and open on the other. A proof that the speeding deliveryman problem on a tree with unit-time windows is NP-hard follows the same form.

7 Summary

In this paper, we have presented $3 + \epsilon$ and $6 + \epsilon$ factor approximation algorithms for the unrooted version of the traveling repairman problem on a tree and on a metric graph, respectively, when all time windows have the same length. We have also posed the speeding deliveryman problem and presented $4 + \epsilon$ and $\frac{20}{3} + \epsilon$ factor approximation algorithms for that problem on a tree and on a metric graph, respectively, with the same restriction. These algorithms rely crucially on our technique of trimming to group time windows into non-overlapping periods.

We have extended our approximation algorithms to both problems when the range of lengths of time windows is bounded by some ratio $D$. In both cases, our approximation ratios are constant for any fixed $D$. For the traveling repairman problem, our approximation improves on the constant given in [7], while for the speeding deliveryman problem, we believe that our results are the first constant-factor approximations. We have also sketched a proof of NP-hardness for the repairman and deliveryman problems on a tree.

Open questions include whether better constant approximations for either problem are possible for the case of bounded length time windows and whether a constant approximation is possible at all when there is no bound on the relative lengths of time windows. Additionally, we can ask what happens if we can deploy more than one repairman or have the option of increasing the speed of his vehicle. We have designed approximation algorithms for multivehicle versions of the repairman problem in [16]. Moreover, we have designed bicriteria approximation algorithms that guarantee a correspondingly higher proportion of the optimal profit as the speed of the repairman’s vehicle is increased over a certain range [14, 15].

Acknowledgements We would like to thank the referees for their careful reading of the manuscript and their many helpful suggestions.

References

1. Arkin, E.M., Mitchell, J.S.B., Narasimhan, G.: Resource-constrained geometric network optimization. In: Proc. 14th Symp. on Computational Geometry, pp. 307–316. ACM, New York (1998)
2. Awerbuch, B., Azar, Y., Blum, A., Vempala, S.: New approximation guarantees for minimum-weight $k$-trees and prize-collecting salesmen. SIAM J. Comput. 28(1), 254–262 (1998)
3. Bansal, N., Blum, A., Chawla, S., Meyerson, A.: Approximation algorithms for deadline-TSP and vehicle routing with time-windows. In: Proc. 36th ACM Symp. on Theory of Computing, pp. 166–174 (2004)
4. Bar-Yehuda, R., Even, G., Shahar, S.: On approximating a geometric prize-collecting traveling salesmen problem with time windows. J. Algorithms 55(1), 76–92 (2005)
5. Blum, A., Chawla, S., Karger, D.R., Lane, T., Meyerson, A., Minkoff, M.: Approximation algorithms for orienteering and discounted-reward TSP. SIAM J. Comput. 37(2), 653–670 (2007)
6. Chawla, S.: Graph algorithms for planning and partitioning. PhD thesis, Carnegie Mellon University (2005)
7. Chekuri, C., Korula, N.: Approximation algorithms for orienteering with time windows (2007). arXiv:0711.4825
8. Chekuri, C., Kumar, A.: Maximum coverage problem with group budget constraints and applications. In: 7th Int. Workshop on Approximation Algorithms for Combinatorial Optimization Problems. LNCS, vol. 3122, pp. 72–83. Springer, Berlin (2004)
9. Chekuri, C., Korula, N., Pál, M.: Improved algorithms for orienteering and related problems. In: Proc. 19th ACM-SIAM Symp. on Discrete Algorithms, pp. 661–670. Society for Industrial and Applied Mathematics, Philadelphia (2008)
10. Chen, K., Har-Peled, S.: The orienteering problem in the plane revisited. In: Proc. 22nd Symp. on Computational Geometry, pp. 247–254. ACM, New York (2006)
11. Focacci, F., Lodi, A., Milano, M.: Solving TSP with time windows with constraints. In: Proc. 1999 Int. Conf. on Logic Programming, pp. 515–529. MIT Press, Cambridge (1999)
12. Focacci, F., Lodi, A., Milano, M.: Embedding relaxations in global constraints for solving TSP and TSPTW. Ann. Math. Artif. Intell. 34(4), 291–311 (2002)
13. Frederickson, G.N., Wittman, B.: Approximation algorithms for the traveling repairman and speeding deliveryman problems with unit-time windows. In: APPROX-RANDOM. LNCS, vol. 4627, pp. 119–133. Springer, Berlin (2007)
14. Frederickson, G.N., Wittman, B.: Speedup in the traveling repairman problem with unit time windows (2009). arXiv:0907.5372
15. Frederickson, G.N., Wittman, B.: Speedup in the traveling repairman problem with constrained time windows (2011). arXiv:1101.3960
16. Frederickson, G.N., Wittman, B.: Two multivehicle routing problems with unit-time windows (2011). arXiv:1101.3953
17. Hoogeveen, J.: Analysis of Christofides’ heuristic: some paths are more difficult than cycles. Oper. Res. Lett. 10(5), 291–295 (1991)
18. Karp, R.M.: Reducibility among combinatorial problems. In: Miller, R.E., Thatcher, J.W. (eds.) Complexity of Computer Computations, pp. 85–103. Plenum, New York (1972)
19. Karuno, Y., Nagamochi, H., Ibaraki, T.: Better approximation ratios for the single-vehicle scheduling problems on line-shaped networks. Networks 39(4), 203–209 (2002)
20. Knuth, D.E.: The Art of Computer Programming, 1st edn. Seminumerical Algorithms, vol. 2. Addison-Wesley, Reading (1969)
21. Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., Schmoys, D.B.: The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization. Wiley, Chichester (1985)
22. Nagarajan, V., Ravi, R.: Poly-logarithmic approximation algorithms for directed vehicle routing problems. In: APPROX-RANDOM. LNCS, vol. 4627, pp. 257–270. Springer, Berlin (2007)
23. Papadimitriou, C.H., Yannakakis, M.: The traveling salesman problem with distances one and two. Math. Oper. Res. 18(1), 1–11 (1993)
24. Tsitsiklis, J.N.: Special cases of traveling salesman and repairman problems with time windows. Networks 22, 263–282 (1992)