Algorithms for Capacitated Vehicle Routing

Moses Charikar [*] Samir Khuller [†] Balaji Raghavachari [‡]

November 20, 1997

Abstract

Given n identical objects (pegs), placed at arbitrary initial locations, we consider the problem of transporting them efficiently to n target locations (slots) with a vehicle that can carry at most k pegs at a time. This problem is referred to as k-delivery TSP, and it is a generalization of the Traveling Salesman Problem. We give a 5-approximation algorithm for the problem of minimizing the total distance traveled by the vehicle.

There are two kinds of transportations possible — one that could drop pegs at intermediate locations and pick them up later in the route for delivery (preemptive) and one that transports pegs to their targets directly (non-preemptive). In the former case, by exploiting the freedom to drop, one may be able to find a shorter delivery route. We construct a non-preemptive tour that is within a factor 5 of the optimal preemptive tour. In addition we show that the ratio of the distances traveled by an optimal non-preemptive tour versus a preemptive tour is bounded by 4.

Keywords: Vehicle routing, Traveling salesman problem, Approximation algorithms, Graphs.

AMS Classification: 90B06, 90C27, 68Q20, 68R10, 05C38, 05C85.

1. Introduction

Vehicle routing and delivery problems have been widely studied in Computer Science and Operations Research. Many of these problems are NP-hard, and a lot of research has been done on analyzing heuristics to find "good" solutions to these problems. These transportation problems occur in real life in areas such as robotics and transportation of packages. Methods for obtaining "good" solutions to the problems are of great practical significance. For example, Casco *et al* [9] report that combining deliveries and pickups for supermarkets led to an industry wide savings of \$160 million a year. The problem that we consider in this paper is that of transporting a single commodity from a set of suppliers to a set of demand points using a vehicle of limited capacity.

[*]Department of Computer Science, Stanford University, Stanford, CA 94305. Supported by an ARO MURI Grant DAAH04-96-1-0007 and NSF Award CCR-9357849, with matching funds from IBM, Schlumberger Foundation, Shell Foundation, and Xerox Corporation. E-mail : moses@cs.stanford.edu.

[†]Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742. Research supported by NSF Research Initiation Award CCR-9307462 and an NSF CAREER Award CCR-9501355. E-mail : samir@cs.umd.edu.

[‡]Department of Computer Science, The University of Texas at Dallas, Richardson, TX 75083-0688. Research supported in part by NSF Research Initiation Award CCR-9409625. E-mail : rbk@utdallas.edu.

One way to analyze the performance of a heuristic is to compute the worst-case ratio between the cost of a solution produced by the algorithm to the cost of an optimal solution. If this ratio is bounded by ρ, we refer to this algorithm as an approximation algorithm with performance ratio ρ or simply as a ρ-approximation algorithm.

k-Delivery Traveling Salesman Problem: Given n identical pegs placed at arbitrary locations, a vehicle with a maximum capacity of k pegs, and n slots (demand points), each requiring a peg, the problem is to find a shortest tour for the vehicle in which all the pegs can be transported to their slots without exceeding the capacity of the vehicle. This problem is referred to as k-delivery TSP. It is a generalization of the traveling salesman problem (TSP), and is therefore NP-hard. The distances between the given points satisfies the triangle inequality since the vehicle can always take a shortest path between any two points. Replacing the distances between each pair of points by the shortest-path distance between them ensures that the triangle inequality is satisfied.

Haimovich and Rinnooy Kan [17] studied a *special case* of this problem when all the pegs are located at one central depot, and are delivered with a vehicle of capacity k. They provided an approximation algorithm that obtains a performance ratio of 3. For geometric graphs — graphs induced by points in the plane with Euclidean distances as edge-weights, they provided a polynomial time approximation scheme for constant k. Christofides [11] surveys various issues, including problem formulation and algorithms, related to the vehicle routing problem, where the vehicles originate at a central depot. In STOC '97, Asano *et al* [6] gave a polynomial-time approximation scheme for the same special case in the geometric setting, when k is $O(\log n/\log\log n)$. Anily and Hassin [2] demonstrated an algorithm that obtains a ratio of 2.5 for the 1-delivery TSP. The first constant factor approximation algorithm for the general problem was given by Chalasani, Motwani and Rao [10]. We will refer to their algorithm as the "CMR algorithm". They obtained an approximation ratio of 9.5. They also gave better algorithms for the cases $k = 1$ and $k = \infty$, that obtain ratios of 2 in both cases. Independent of our work, Anily and Bramel [1] showed that a modification of the CMR algorithm improves the approximation ratio to $7 - \frac{3}{k}$ (in fact, we show that one can obtain a better bound of 6.5 by modifying the CMR approach). They also gave another algorithm with an approximation ratio of $c(k) + \frac{1}{2}\log_2 k$, where $2 \leq c(k) \leq 3$.

Our Results: We summarize the results presented in this paper below:

- For the k-delivery TSP, we provide a natural approximation algorithm that runs in polynomial time and show that its performance ratio is at most 5. Since the proof is complex, we first prove a simpler bound of 6.5; this proof contains some of the basic ideas.
- For geometrical instances, such as points in the plane, the algorithms of Arora [4, 5] and Mitchell [20]) can be used to obtain an $(1 + \epsilon)$ approximation of the TSP, and this leads to an approximation factor of $4(1 + \epsilon)$ for these instances.
- We also describe a *simple* algorithm that finds a preemptive tour whose length is at most 5 times the length of an optimal preemptive tour.

Significance of our work: We explain below how our algorithm fundamentally differs from previous algorithms and why it is likely to return far better solutions in practice than them. The previous methods for solving the general k-delivery TSP suffer from the following drawback. They start with two tours, one containing all the pegs (source nodes) and the other containing all the slots (delivery nodes). The basic idea is to traverse the cycle of pegs, collecting k pegs, then switch over to the other cycle and deliver the k pegs, repeating the process until all pegs are delivered. The delivery route thus alternates between the two cycles; it turns out that it is easy to analyze the cost of shuttling between the cycles in this scheme. Such a scheme suffers from the drawback that the individual tours for the pegs and slots do not take advantage of the proximity of pegs and slots. In most instances, one can do better by alternating pickups and deliveries without waiting for the vehicle to become either completely full or completely empty. We derive a natural algorithm that *uses a single tour* containing all points, combines pickups and deliveries arbitrarily, and takes "corrective" action only when the vehicle becomes either full or empty. The main hurdle is in proving a good analysis of this more natural scheme. Our analysis shows significantly better approximation ratios for our algorithm than the previous algorithms. In addition, preliminary experimental studies show that our algorithm returns much better solutions.

Note: We will assume that we can start the vehicle's tour at any location. If the starting point is fixed, our method still applies with an additive factor of 1 in the approximation factors.

Preemptive Tours: A fundamental issue is that of *preemptive* versus *non-preemptive* traversals. In a preemptive traversal, pegs may be dropped at intermediate locations; in other words, we may pick up a peg and leave it at some location, and return later to collect it and deliver it. In a non-preemptive solution, we carry a peg from its source to its destination without ever unloading it from the vehicle at intermediate nodes. The nature of the problem and algorithms to find solutions can change if drops are permitted.

Fig. 1 shows an example in which the best preemptive tour is shorter than the best non-preemptive tour. The edges shown cost 1 unit each. A tour with a capacity-2 vehicle that leaves a peg at point A and delivers it later costs 8 units. If we are not allowed to drop a peg at intermediate locations, we incur a higher cost (regardless of where the tour starts), and an optimal non-preemptive tour costs 10 units. This raises a very fundamental question — *what is the worst-case ratio of the cost of an optimal non-preemptive tour to that of an optimal preemptive tour?* Our example shows that the ratio is at least $\frac{5}{4}$.

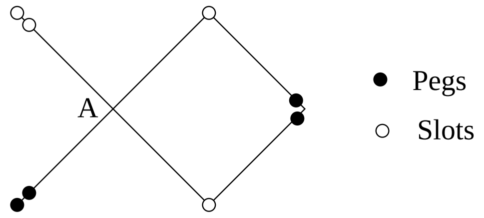

Figure 1: Example to show that intermediate drops help even for $k = 2$.

We show that the ratio between the optimal non-preemptive and preemptive tours is at most 4, by showing that given a preemptive tour of length L, we can find a non-preemptive tour of length at most $4L$. This theorem is proven by using a variety of different ideas. One interesting method is a general technique for simulating a preemptive tour of a unit capacity vehicle by a non-preemptive tour, *that travels the same distance* (Lemma 2.7). We also develop many other methods that are needed in showing that the constants are fairly small — one idea is to collect $\frac{k}{2}$ pegs and to deliver them, rather than waiting to collect k pegs.

Related Work: A closely related problem is the *stacker-crane* problem. This problem too is that of making deliveries with a vehicle of capacity k. But, in the stacker-crane problem, the objects are *not* identical and each object has a specific target destination. The goal is to find a shortest tour that performs the transportation. For the unit capacity case, Frederickson, Hecht and

Kim [15] gave an algorithm with an approximation factor of 1.8. For the case when the underlying metric is a tree, Frederickson and Guan [13, 14] have given fast algorithms to compute optimal solutions for the preemptive case, and fast approximation algorithms for the non-preemptive case (the problems are NP-hard even for trees). Knuth [19, Section 5.4.8] discusses Karp's work [18] on the problem for paths and trees. Fast algorithms were given by Atallah and Kosaraju [7] for the cases in which the graph is either a simple cycle or path. The algorithms are slightly faster for the cases when drops are permitted. Frederickson [12] showed improved running times for a cycle when no drops are allowed. The issue of tours under various types of restrictions has also been investigated by Arkin, Hassin and Klein [3].

Outline of the paper: In Section 2 we describe an approximation algorithm for the k-delivery TSP that obtains an approximation ratio of 5. We first prove a simpler bound of 6.5 in Section 2, and then in Section 3, we provide a better analysis of our algorithm. We also show how to "convert" a preemptive tour to a non-preemptive tour. We prove that the total length of the non-preemptive tour obtained by our algorithm is at most 4 times the length of the preemptive tour. In Section 4, we describe a simple algorithm that finds a preemptive tour whose length is at most 5 times the length of an optimal preemptive tour.

Notation: An optimal non-preemptive tour for the k-delivery TSP is denoted by C_k, and an optimal preemptive tour is denoted by C'_k. We will use C_k or C'_k to denote the length of the tour as well, and one can distinguish between the two meanings from the context.

2. An approximation algorithm for k-delivery TSP

In this section, we provide an approximation algorithm and an analysis for its performance. We also show how to simulate a preemptive tour by a non-preemptive tour. In particular, we show that $C_1 \leq kC'_k$ (Lemma 2.7). (This also shows the interesting result that $C_1 = C'_1, C_2 \leq 2C'_2, C_3 \leq 3C'_3$.)

We prove the following theorems.

Theorem 2.1: *Consider an arbitrary instance of k-delivery TSP. There is a polynomial-time approximation algorithm that finds a non-preemptive tour whose length is at most 5 times the length of an optimal tour (possibly preemptive). In the special case when the points are specified on the plane, and edge-costs are specified by Euclidean distances, the approximation ratio can be improved to $4(1+\epsilon)$, for any constant $\epsilon > 0$.*

Theorem 2.2: *The length of an optimal non-preemptive tour of a k-delivery TSP instance is at most 4 times the length of an optimal preemptive tour, i.e., $C_k \leq 4C'_k$.*

2.1. Overview of the algorithm

The main idea is the following: first construct a tour of *all* the given points. Starting from some initial vertex, we traverse the tour, picking up pegs, and delivering them to slots, on-line. In other words, when the vehicle passes through a node with a peg, it picks up the peg, and when it passes through a slot, it drops a peg there. We show that if the vehicle has unbounded capacity, then there is always a starting point such that the vehicle can complete all deliveries without ever running out of pegs. If the vehicle has bounded capacity k, the simple scheme outlined above does not work directly. We need to address the following two situations: (a) the vehicle is full when a peg is visited, and (b) the vehicle is empty when a slot is visited.

In the following discussion, we assume that k is even. We will describe how to handle the case when k is odd later. The performance ratio is at most $5\frac{1}{6}$ for odd k. We treat the vehicle as full when it has $k/2$ pegs on it, and the remaining capacity is used as a "buffer". The tour is broken up into segments of 3 kinds: (i) *neutral* segments with equal number of pegs and slots, (ii) *positive* segments that have $k/2$ more pegs than slots, and (iii) *negative* segments that have $k/2$ more slots than pegs. Neutral segments are processed as mentioned in the above scheme. In most practical situations, most parts of the tour constructed may be neutral segments, and in this case our algorithm would do very well since neutral segments are processed by traversing them only once. By definition, there are as many positive segments as negative segments. We compute a minimum-weight perfect matching between the positive and the negative segments. When the vehicle is passing through a positive segment on its tour, it delivers the excess pegs to the negative segment to which the positive segment is matched.

The main difficulty in analyzing such a scheme is that the cost of a matching between the positive and the negative segments has to be bounded with respect to an optimal tour. Note that the matching does not include all nodes in the original problem and therefore it could possibly be arbitrarily expensive. Another complication is that the segments do not have the same number of points on them. The techniques used in the previous results [1, 10] do not yield a bound on the cost of such a matching. We show how to bound the cost of the matching with respect to an optimal tour, and use it to derive a better approximation bound. We now describe the algorithm in detail.

2.2. The algorithm

1. Construct a tour T that visits all the points.
2. Fix a reference point P on the tour T.
3. Traverse the tour T in some direction starting from P.
4. Compute the *excess* function $\textsc{excess}(e)$ for each edge e of T. $\textsc{excess}(e) = \textsc{pegs}(e) - \textsc{slots}(e)$, where $\textsc{pegs}(e)$ is the number of pegs encountered before e is traversed, and $\textsc{slots}(e)$ is defined analogously as the number of slots encountered before e.
5. For each value of $i \in [0, k/2)$ do
   (a) Break the tour into pieces by removing all edges with $\textsc{excess}(e) \equiv i \bmod k/2$. Call these edges *cut* edges. Fig. 2 shows a sample tour and a plot of the excess function for a counterclockwise traversal of the tour (with $k = 6$ and $i = 0$).
   (b) We get p-pieces (positive pieces), n-pieces (negative pieces) and 0-pieces (zero or neutral pieces) as follows.
      - A p-piece is one where the *excess* function is x on the cut edge preceding the piece and $x + k/2$ on the cut edge following the piece; for all edges e in the piece, $\textsc{excess}(e) \in (x, x + k/2)$.
      - A n-piece is one where the *excess* function is x on the cut edge preceding the piece and $x - k/2$ on the cut edge following the piece; for all edges e in the piece, $\textsc{excess}(e) \in (x - k/2, x)$.

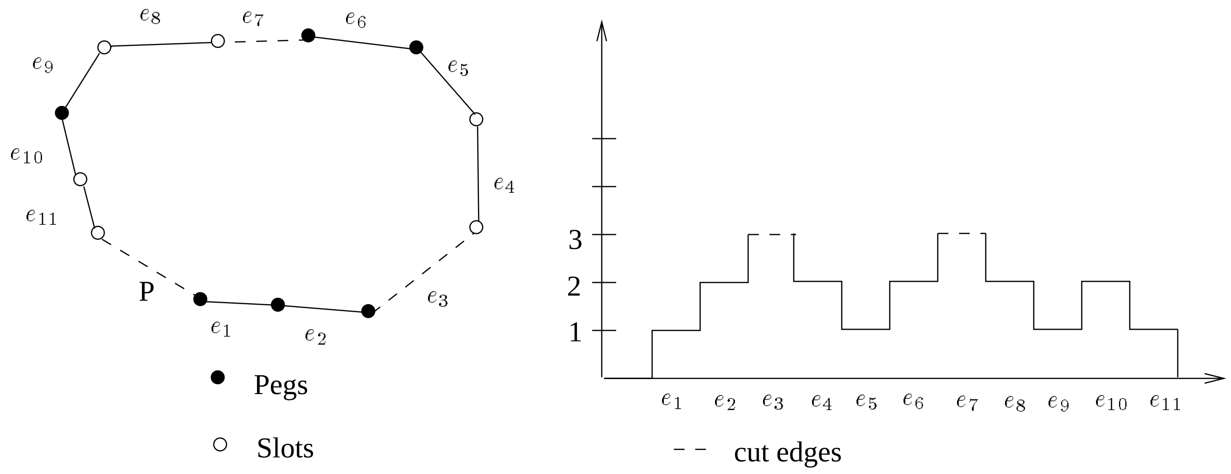

Figure 2: A tour and the excess function plotted for $k = 6$ and $i = 0$.

- A 0-piece is one where the *excess* function is x on the cut edge preceding the piece and x on the cut edge following the piece; for all edges e in the piece, either $\text{EXCESS}(e) \in (x - k/2, x)$ (such a piece is called *decreasing*) or $\text{EXCESS}(e) \in (x, x + k/2)$ (such a piece is called *increasing*).

(c) Compute a matching on the p- and n-pieces as follows:

- Construct a weighted bipartite graph $B = (V^+, V^-, E)$ on p- and n-pieces as follows:
- There is one vertex in V^+ for each p-piece and one vertex in V^- for each n-piece. For each p-piece u^+ and each n-piece w^-, the edge (u^+, w^-) has weight equal to the minimum-weight edge connecting a vertex in u^+ and a vertex in w^-.
- Compute a minimum-weight perfect matching M in the bipartite graph B. A p-piece and a n-piece that are matched to each other are said to form a p/n-pair.

(d) Now traverse the tour starting from any point in both clockwise and anticlockwise directions as follows: (Assume for now that we start each traversal at the beginning of a piece with exactly $k/2$ pegs. Lemma 2.3 shows how this assumption is unnecessary).

- On encountering a 0-piece, we move along the piece picking up pegs and delivering them. Since we start with $k/2$ pegs, we can do this in a single traversal and reach the end of the piece with $k/2$ pegs. Note that we leave the piece with $k/2$ pegs.
- When we encounter the first piece of a p/n-pair, we service the pair as follows: suppose we encounter a p-piece P^+ which is matched with n-piece P^- by edge $e \in M$. Note that there must be an edge e' corresponding to e, such that e' connects some vertex in P^+ to some vertex in P^-, and the weight of e' is the same as that of e. Traverse P^+, performing pickups and deliveries, until e' is encountered. Now, move to the beginning of P^-, and traverse P^- performing pickups and deliveries. Then move back to the point in P^+ where we left off and continue performing pickups and deliveries. Note that after servicing a p/n-pair thus, we finish with $k/2$ pegs. The case when the n-piece is encountered before the p-piece is handled similarly.
- When we encounter the second piece of a p/n-pair, we simply traverse the piece without performing any pickups and deliveries (since it has already been serviced).

(e) For both the tours (clockwise and anticlockwise) described above, find a valid starting point (i.e., a starting point such that we never run out of pegs and the number of pegs carried is at most k). Lemma 2.3 guarantees the existence of such a starting point.

6. Return a shortest tour from amongst the k tours constructed (two for each $i \in [0, k/2)$).

2.3. Analysis: a weaker bound

We can prove an upper bound of 5 on the approximation ratio achieved by the above algorithm. Since the proof is complex, we prove a simpler bound of 6.5 that contains the basic ideas. In fact, for the simpler proof we can fix $i = 0$ in the above algorithm. To prove the better bound, we need to try all values of i. Section 3 contains the proof for the bound of 5. We also prove the relationship between the tour generated by the algorithm and C'_k (Theorem 2.2).

Lemma 2.3: *A valid starting point is guaranteed to exist in Step 5e of the algorithm.*

Proof: Suppose we started the tour constructed by the algorithm at the beginning of any piece with a vehicle preloaded with $k/2$ pegs. Then the following invariant is maintained throughout the tour: The vehicle has exactly $k/2$ pegs when it traverses a cut edge from one piece to another. We can verify that the number of pegs carried by the vehicle always lies in the interval $[0, k]$. This would be a valid tour except for the fact that we assumed that we started with $k/2$ pegs initially. Let $n(e)$ be the number of pegs carried by the vehicle as it traverses edge e (beginning with $k/2$ pegs). Consider the edge $e_{\min}$, where $n(e)$ reaches its minimum value, say x (break ties arbitrarily). Suppose we start a new vehicle with no pegs from edge $e_{\min}$. Let $n'(e)$ be the number of pegs carried by the new vehicle as it traverses edge e. It can be verified that $n'(e) = n(e) - x$. By the choice of $e_{\min}$, this ensures that the number of pegs carried by the vehicle always lies in the interval $[0, k]$. This proves the existence of a valid starting point for the tour and in fact, also gives a simple method to find such a starting point. □

Based on the above lemma, we can also show the following.

Lemma 2.4: *There is a polynomial-time approximation algorithm for the ∞-delivery TSP (i.e., the vehicle has infinite capacity) with a performance ratio of 1.5. For geometric instances on the plane, the approximation ratio is $1 + \epsilon$, for any $\epsilon > 0$.*

Proof: Let T_{OPT} be a minimum length tour of all the points. Since C'_k is a tour of the points, $T_{OPT} \leq C'_k$. The algorithm constructs a tour T of all the points. For the algorithm, we will assume that an α-approximation of TSP tour is used, and therefore the weight of the tour is at most $\alpha T_{OPT} \leq \alpha C'_k$. If Christofides' algorithm is used, then $\alpha \leq 1.5$. For geometric instances, such as points in the plane, the algorithms of Arora [4, 5] and Mitchell [20] can be used to obtain a $(1 + \epsilon)$ approximation of the TSP tour. In this case, $\alpha = 1 + \epsilon$, for any constant $\epsilon > 0$. Using the same ideas as in the above lemma we can show that there is always a valid starting point on this tour, such that we never run out of pegs. □

In Step 5c we find a matching M on the p- and n-pieces. We need to bound the weight of the matching M. We cannot bound the weight of the matching by the method used in [10] since the matching is not being found on a graph that involves all the pegs and slots. We use a different approach to bound the weight of the matching.

Let A be a minimum-weight perfect matching between pegs and slots, where a peg must be matched to a slot. We use A to denote the matching, and its weight. One can distinguish between the two meanings from the context.

2.3.1. Bounding the matching A

The following lemmas derive an upper bound on the ratio of the weight of matching A to the weight of an optimal preemptive solution. Even though there are other ways of proving this directly, these proof methods also show how to "simulate" a preemptive solution with a non-preemptive solution.

Our main goal is to prove the following theorem.

Theorem 2.5: $A \leq \frac{k}{2} C'_k$.

This theorem follows once we establish the following lemmas.

Lemma 2.6 (Chalasani, Motwani and Rao [10]): $A \leq \frac{1}{2} C_1$.

We now prove that a non-preemptive unit-capacity vehicle can simulate a preemptive capacity-k vehicle, with an increase of the length of the tour by a factor of k. One could attempt to prove this by walking around the cycle k times, but the problem is that we may attempt to pick up a peg that is not yet "available".

Lemma 2.7: $C_1 \leq k \cdot C'_k$.

Proof: Consider a tour C'_k which delivers pegs to slots and is allowed intermediate dropping points. We will show that we can convert this tour into one of length $k \cdot C'_k$ where the pegs are carried to slots with no intermediate drops by a unit capacity vehicle. Assume the vehicle of capacity k starts at s and returns to s.

We construct an auxiliary multigraph $G = (V, E)$ from the tour as follows. The vertex set of graph is defined to be:

$$V = \{s\} \cup \{x | x \text{ is an intermediate drop point }\}$$

Imagine that we store the pegs on numbered compartments in the vehicle. The numbers are $1 \ldots k$. The tour starts from s and visits pegs/slots and intermediate drop points. We can view the tour as "segments" where it goes back and forth between the vertices of the graph we constructed (see Fig 3). We create k edges in G for each movement done by the vehicle between two vertices of G. Each edge corresponds to a numbered compartment. We associate with each such edge, the pegs which were placed into this compartment and the slots which were serviced by removing a peg from this compartment. Thus each edge is associated with an alternating sequence of pegs and slots. As the vehicle moves from vertex x to vertex y of G, each numbered compartment undergoes various changes:

1. Lose a peg (leave x with a peg, arrive at y with no peg). The corresponding edge is associated with a peg/slot sequence of the form $sps \ldots ps$. (Here p represents a peg and s a slot).

2. Gain a peg (leave x with no peg, arrive at y with a peg). The corresponding edge is associated with a sequence of the form $psp \ldots sp$.
3. Move without carrying anything from x to y (we may load and unload pegs in this compartment during the motion). The corresponding edge is associated with a sequence of the form $psp \ldots ps$ (possibly empty).
4. Move carrying a peg from a vertex to another vertex (we may unload and load pegs in this compartment during the motion). The corresponding edge is associated with a sequence of the form $sps \ldots sp$ (possibly empty).

Each edge corresponding to a compartment is a labeled edge in the multigraph. Type (1) edges are labeled as $-$ edges. Type (2) edges are labeled as $+$ edges. We subdivide edges of type (3) by a vertex xy. The edge from x to xy gets the label $+$ and the edge from xy to y is labeled as a $-$ edge. We subdivide edges of type (4) by a vertex xy. The edge from x to xy gets the label $-$ and the edge from xy to y is labeled as a $+$ edge. We can think of a $+$ edge as being associated with an odd length alternating peg/slot sequence that begins and ends with a peg. Similarly, a $-$ edge can be thought of as being associated with an odd length alternating peg/slot sequence that begins and ends with a slot.

Lemma 2.8: *For each vertex of the auxiliary multigraph G, the number of $+$ labels incident to it is equal to the number of $-$ labels incident to it.*

Proof: For each compartment, the contribution to the $+$ and $-$ label is the same. Consider a vertex v that is a dropping point. The number of pegs that are carried to it are carried away. (Formally, each vertex v occurs a number of times on the tour. There are 4 cases (a) come with a peg, leave it here and go $(+v+)$, (b) come with a peg, leave with a peg $(+v-)$, (c) come with no peg, go with no peg $(-v+)$, (d) come with no peg, go with a peg $(-v-)$. Number of case (a) and (d) are the same since (a) increases peg count and (d) decreases peg count.) $\square$

In order to construct a solution that does not drop pegs at intermediate locations, we need to construct an alternating sequence of pegs and slots which visits every peg and every slot. Find an Euler tour that alternates using $+$ and $-$ edges in this multigraph. (When we enter a vertex on a $+$ edge we can leave on a $-$ edge and vice-versa.) The Euler tour can be interpreted as a non-drop tour solution for a vehicle with unit capacity. The unit capacity vehicle simply traverses the edges in the order of the Euler tour. When we traverses a particular edge, we service the sequence of pegs and slots associated with that edge. The definition of $+$ and $-$ edges ensures that we encounter pegs and slots in an alternating fashion. This therefore gives a non-drop tour solution with unit capacity. This completes the proof of Lemma 2.7. $\square$

We illustrate this construction by an example in Figure 3. Suppose we are given the preemptive tour that starts at s, pick up a peg on path A and drop it off at the first drop off point. We then take the loop marked B performing one delivery, come back to pick up the dropped off peg and take the loop marked C. We return with a peg, drop it off at the same place, we then take the path D, drop off a second peg, come back to pick up the first peg on path E, deliver it on path F, pick up the second peg and take path G, returning to s.

Corresponding to this traversal, we construct the auxiliary multigraph G. We have three vertices to begin with. Path A gains a peg and is marked $+$. Path B is a loop on which we leave without a peg and return without a peg, we subdivide this edge and it is a $+-$ edge. Path C is a loop on which we leave with a peg and return with a peg and so we subdivide it and mark it a $-+$ edge. In a similar manner we finish the construction of the multigraph. An Euler tour in this multigraph that alternates between + and - edges is easy to find. This corresponds to a non-preemptive traversal.

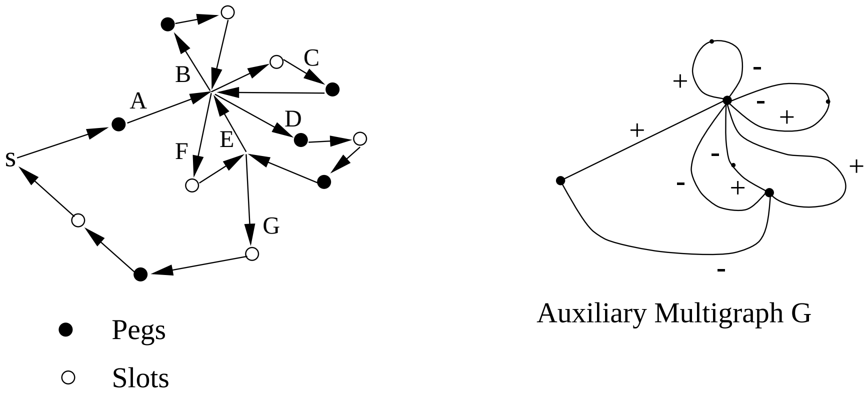

Figure 3: Illustrating construction of auxiliary multigraph $G = (V, E)$ for a unit capacity vehicle.

2.3.2. Bounding the matching M

Let W be a matching (which we call a *wiggly matching*) that matches pegs to slots such that all points are matched except for some $k/2$ pegs in each p-piece and $k/2$ slots in each n-piece. We use the *wiggly matching* to bound the cost of the matching M as follows:

Lemma 2.9:

$$M \leq \frac{2}{k}(A + W) \tag{1}$$

Proof: Consider the symmetric difference of A and W. A is a perfect matching and W is a matching where all the pegs and slots in the 0-pieces are matched, and in each p- (n-) piece, there are $\frac{k}{2}$ unmatched pegs (slots). The symmetric difference has even length cycles and paths. Since A is a perfect matching, the symmetric difference is a collection of disjoint augmenting paths (with respect to W) that start and end with edges from A at vertices of degree 1 in the graph $A \cup W$. The only vertices having degree 1 are pegs in the p-pieces and slots in the n-pieces. Each augmenting path with respect to W has a peg at one end and a slot at the other end. This gives us paths from pegs to slots such that each p-piece has $k/2$ paths emerging from it and each n-piece has $k/2$ paths ending on it. This collection of paths can be decomposed into $k/2$ perfect matchings between p- and n-pieces(because the edges of a d-regular bipartite graph can be decomposed into d perfect matchings [8]). It follows that the weight of the minimum weight perfect matching on the p- and n-pieces is at most the average weight of the $\frac{k}{2}$ matchings we found. By triangle inequality, the weight of the symmetric difference is at most the sum of the weights of the matchings. This proves the lemma. □

Recall that T is the tour of all the points constructed by the algorithm. Let $T^{(\pm)}$ be the total length of the p/n-pieces and let $T^{(0)}$ be the total length of the 0-pieces. Let $T^{(C)}$ by the total length of the cut edges. Then $T = T^{(\pm)} + T^{(0)} + T^{(C)}$.

2.3.3. Bounding the matching W

Lemma 2.10:

$$W \leq \frac{k}{2}\left(\frac{1}{2}T^{(\pm)} + T^{(0)}\right) \tag{2}$$

Proof: We will construct the *wiggly matching* separately for 0- and p/n-pieces. A 0-piece can be traversed in one of the two directions by a vehicle with no pegs initially such that pegs are picked up and delivered without having to back up. Suppose this vehicle operates like a stack, pushing and popping pegs. For each $j \in [0, k/2)$, consider the pegs which were placed in location j of this stack and the slots which were serviced by popping a peg from location j of this stack. This defines a path of the form $psp\ldots s$ where p stands for a peg and s stands for a slot. We match each peg on this path with the slot immediately following it. The total length of the matching edges is at most the length of the path. But the length of the path is at most the length of the 0-piece. Since we construct $k/2$ paths, the total length of the matching within this 0-piece is at most $k/2$ times the length of the 0-piece.

A p-piece can be traversed (in either direction) by a vehicle with no pegs initially such that pegs are picked up and delivered without having to back up. The vehicle ends up with $k/2$ pegs at the end. Analogous to the construction for a 0-piece, we define a path corresponding to each location j of the stack. Each path is of the form $psp\ldots sp$. Now we can match pegs with slots either starting from the left or the right. In either case, one peg is left unmatched. We choose the lower cost matching amongst the two. The total cost of the two matchings is exactly the cost of the path. Thus the smaller of the two matchings costs at most half the length of the path. Again the length of each path is at most the length of the p-piece. Since we construct $k/2$ paths, the total length of the matching within this p-piece is at most $k/4$ times the length of the p-piece. A similar construction can be done for n-pieces. Hence the lemma follows. □

Theorem 2.5 bounds the weight of the matching A to be at most $\frac{k}{2}C'_k$. Substituting into (1) for A from the theorem and for W from (2), we get

$$M \leq C'_k + \frac{1}{2}T^{(\pm)} + T^{(0)} \tag{3}$$

2.3.4. Proof of Theorems 2.1 and 2.2.

Note that in the two tours constructed by the algorithm, each 0-piece is traversed once and each cut edge is traversed once. Also the first of each p/n-pair is traversed once while the second is traversed thrice. Averaging over the two traversals, we charge each p/n-piece twice. Each edge in M is traversed twice.

Hence the average length of the two tours constructed is at most

$$\begin{aligned} 2M + T^{(0)} + 2T^{(\pm)} + T^{(C)} &\leq 2C'_k + 3T^{(0)} + 3T^{(\pm)} + T^{(C)} \\ &\leq 2C'_k + 3T \\ &\leq 2C'_k + 3\alpha C'_k = (2+3\alpha)C'_k \end{aligned}$$

Since $\alpha \leq 1.5$, we obtain a worst case approximation ratio of 6.5. We can also view this as a constructive proof bounding the ratio of C_k to C'_k. In that case $\alpha = 1$ and we obtain an upper bound of 5 on the ratio. In Section 3 we show a bound of $(2 + (2 - \frac{4}{k^2})\alpha)C'_k$. Using this improved bound, we get an approximation ratio of 5, and we can bound the ratio of C_k to C'_k by 4. Also, for geometric instances (consisting of points in the plane with Euclidean distances), using an $(1+\epsilon)$-optimal tour [4, 5, 20] in the first step of the algorithm, we obtain a ratio of $4(1+\epsilon)$. □

If we are given a fixed starting point q, we obtain a tour that starts and ends at q as follows. We apply the algorithm as usual. Suppose this constructs a tour that starts and ends at q', which is a peg without loss of generality. We first move from q to q', traverse the tour constructed by the algorithm and on reaching q' again, move back to q. We pay an additional cost of twice the distance between q and q', which is at most the cost of the optimal tour, since the optimal tour includes 2 paths between q to q'. This adds 1 to our approximation ratios.

3. A better analysis

Notice that the previous analysis did not use the fact that we try different break-points and average over them. We now present a better analysis that builds on the basic ideas from Section 2.3 to show that the average tour length for the k tours constructed is actually at most $(2+(2-\frac{4}{k^2})\alpha)C'_k$.

Examining the proof of Lemma 2.10, we observe that the proof constructs $k/2$ paths within each piece of the tour. The contribution to the total length of the final tour is the average path-length in the p/n-pieces plus twice the average path-length in the 0-pieces. Here, "average" refers to the average length of the $k/2$ paths. The earlier proof bounded each path-length by the length of the entire piece. We will improve the analysis by getting better estimates for the average path-lengths by averaging over all values of i used to define the cut edges of the tour.

For a node v (peg or slot) in the tour T, define $\textsc{before}(v)$ to be the edge preceding v in T and $\textsc{after}(v)$ to be the edge following v in T.

It will be useful to identify the paths constructed by the proof of Lemma 2.10 with the values of the *excess* function on edges within a piece. In the following discussion, we fix a value of i used to define the cut-edges. For a given i, for every integer $j \in [0, k/2)$, we construct a path in each piece based on the values of the *excess* function as follows. An edge e is said to be a j-edge if $\textsc{excess}(e) \equiv j \pmod{k/2}$.

For a p/n-piece, the path consists of the pegs p such that $\textsc{before}(p)$ is a j-edge and the slots s such that $\textsc{after}(s)$ is a j-edge. For a 0-piece, we define the paths differently depending on whether the piece is *increasing* or *decreasing*. For a *increasing* 0-piece, the path consists of the pegs p such that $\textsc{after}(p)$ is a j-edge and the slots s such that $\textsc{before}(s)$ is a j-edge. For a *decreasing* 0-piece, the path consists of the slots s such that $\textsc{after}(s)$ is a j-edge and the pegs p such that $\textsc{before}(p)$ is a j-edge. See Figure 4 for an illustration of this definition of paths. Observe that by this definition, for all cases, the path for $j = i$ always has zero length. For a p-piece, it consists of a single peg. For a n-piece, it consists of a single slot. For a 0-piece, the path for $j = i$ is empty. The reader can verify that the paths constructed by this definition are identical to the paths constructed in the proof of Lemma 2.10.

Let $l_{ij}^{(\pm)}$ be the total length of the paths corresponding to j (by the above correspondence) within the p/n-pieces when i is used to define the cut edges. We define $l_{ij}^{(0)}$ similarly for the paths within the 0-pieces.

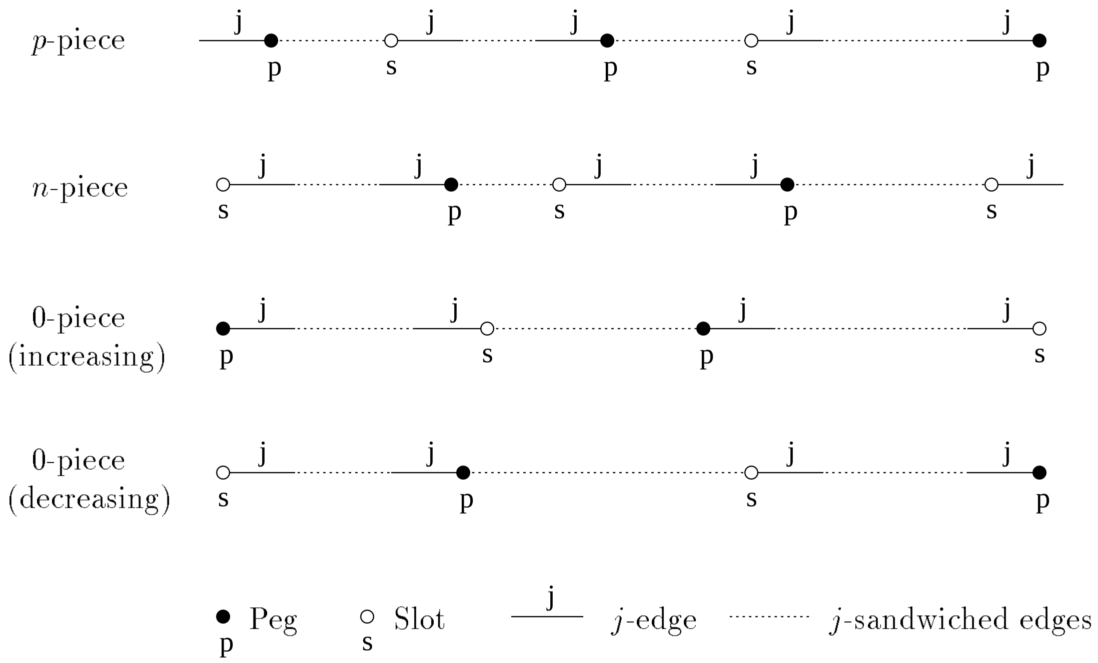

Figure 4: Illustrating the definition of paths in terms of the *excess* function

Call an edge e in a piece j*-sandwiched* if e is not a j-edge and there is some j-edge before e and some j edge after e in the piece (the j-edges need not be immediately before and after e). Note that the property of being a j-edge is independent of the value of i; however, the property of being a j-sandwiched edge depends on the value of i used to define the cut-edges. In any piece, the path corresponding to j (described above) consists of j-sandwiched edges and j-edges Thus the length of this path is bounded by the sum of the total length of the j-sandwiched edges and the total length of the j-edges in the piece.

Let $p_{ij}^{(\pm)}$ be the total length of the j-sandwiched edges within the p/n-pieces when i is used to define the cut edges. We define $p_{ij}^{(0)}$ similarly for the the 0-pieces. Let $q_{ij}^{(\pm)}$ be the total length of the j-edges within the p/n-pieces when i is used to define the cut edges. We define $q_{ij}^{(0)}$ similarly for the 0-pieces. By the above definitions, we have

$$l_{ij}^{(\pm)} \leq p_{ij}^{(\pm)} + q_{ij}^{(\pm)}$$
$$l_{ij}^{(0)} \leq p_{ij}^{(0)} + q_{ij}^{(0)}$$

Also, when i is used to define the cut edges, let $T_i^{(\pm)}$ be the total length of the p/n-pieces, let $T_i^{(0)}$ be the total length of the 0-pieces and let $T_i^{(C)}$ be the total length of the cut edges.

Note that

$$\sum_{j \neq i} q_{ij}^{(\pm)} = T_i^{(\pm)}$$
$$\sum_{j \neq i} q_{ij}^{(0)} = T_i^{(0)}$$

Now, the analysis of the previous section bounded the average length of the two tours produced when a particular value of i is used to define the cut edges. In terms of the notation introduced, this bound can be written as

$$\begin{aligned}
& \frac{4}{k}A + \frac{2}{k}\sum_{j\neq i} l_{ij}^{(\pm)} + \frac{4}{k}\sum_{j\neq i} l_{ij}^{(0)} + 2T_i^{(\pm)} + T_i^{(0)} + T_i^{(C)} \\
\leq \; & 2C'_k + \frac{2}{k}\sum_{j\neq i}\left(p_{ij}^{(\pm)} + q_{ij}^{(\pm)}\right) + \frac{4}{k}\sum_{j\neq i}\left(p_{ij}^{(0)} + q_{ij}^{(0)}\right) + 2T_i^{(\pm)} + T_i^{(0)} + T_i^{(C)} \\
= \; & 2C'_k + \frac{2}{k}\sum_{j\neq i} p_{ij}^{(\pm)} + \frac{4}{k}\sum_{j\neq i} p_{ij}^{(0)} + \left(2 + \frac{2}{k}\right)T_i^{(\pm)} + \left(1 + \frac{4}{k}\right)T_i^{(0)} + T_i^{(C)}
\end{aligned}$$

Let L be the length of the constructed tour averaged over all the $k/2$ values of i. Summing up the above bound over all values of i, we get

$$\frac{k}{2}L \;\leq\; kC'_k + \frac{2}{k}\underbrace{\sum_i\sum_{j\neq i} p_{ij}^{(\pm)}}_{S_1} + \frac{4}{k}\underbrace{\sum_i\sum_{j\neq i} p_{ij}^{(0)}}_{S_2} + \left(2 + \frac{2}{k}\right)\underbrace{\sum_i T_i^{(\pm)}}_{S_3} + \left(1 + \frac{4}{k}\right)\underbrace{\sum_i T_i^{(0)}}_{S_4} + \underbrace{\sum_i T_i^{(C)}}_{S_5} \quad (4)$$

We will compute a bound for sum of the last five terms of the right hand side of (4) in terms of T.

Consider a particular edge e of the tour. Let us calculate its contribution to the various terms in (4). Suppose e is an r-edge. For each value of $i \in [0, k/2), i \neq r$ we scan to the left and right of e for the closest i-edges before and after e. Let A_i be the portion of the path between (and not including) these two i-edges. When i is chosen to define the cut edges of the tour, A_i is a (p/n or 0) piece. Accordingly, we label the segment A_i as p/n or 0. Note that we use the term segment as opposed to piece, as a segment A_i may or may not be a piece depending on the value used to define the cut edges.

Edge e contributes to $l_{ij}^{(\pm)}$ iff A_i is a p/n segment and e is j-sandwiched within A_i, i.e., iff A_j is completely contained within A_i. The edge e contributes to $l_{ij}^{(0)}$ iff A_i is a 0-segment and e is j-sandwiched within A_i, i.e., iff A_j is completely contained within A_i. The edge e contributes to $T_i^{(\pm)}$ iff A_i is a p/n segment and contributes to $T_i^{(0)}$ iff A_i is a 0-segment. It contributes to $T_i^{(C)}$ iff $i = r$.

Claim: If A_i is a p/n segment, it cannot be contained in any other segment A_j $(j \neq i)$.

Proof: Since A_i is a p/n segment, the *excess* function changes by $k/2$ from one end to the other. Hence there must be some point within it where the *excess* function is j mod $k/2$. Hence one of the end points of A_j must be within A_i, proving the claim. □

Suppose x of the segments A_i are 0-segments and $(\frac{k}{2} - x - 1)$ of them are p/n segments. Let us calculate the number of times edge e is counted in the various terms in (4).

The contribution to S_1 is the number of pairs (A_j, A_i) such that A_j is contained in A_i and A_i is a p/n segment. By the above claim, A_j must be a 0-segment. Thus the number of such pairs is at most $x(\frac{k}{2} - x - 1)$. The contribution to S_2 is the number of pairs (A_j, A_i) such that A_j is contained in A_i and A_i is a 0-segment. By the above claim, A_j must be a 0-segment. Note that if A_j is contained in A_i then A_i is not contained in A_j. Thus the number of such pairs is at most

$\frac{x(x-1)}{2}$. The contribution to S_3 is $\frac{k}{2} - x - 1$. The contribution to S_4 is x. The contribution to S_5 is 1.

Thus the contribution of edge e to the sum of the last five terms in (4) is at most

$$\begin{aligned} & \frac{2}{k}x\left(\frac{k}{2} - x - 1\right) + \frac{4}{k}\frac{x(x-1)}{2} + \left(2 + \frac{2}{k}\right)\left(\frac{k}{2} - x - 1\right) + \left(1 + \frac{4}{k}\right)x + 1 \\ = \quad & k - \frac{2}{k} - \frac{2x}{k} \leq k - \frac{2}{k} \end{aligned}$$

Hence the sum of the last five terms in (4) is at most $(k - \frac{2}{k})T$. Using this bound in (4), we get

$$\frac{k}{2}L \leq kC'_k + \left(k - \frac{2}{k}\right)T$$

and hence

$$\begin{aligned} L & \leq 2C'_k + \left(2 - \frac{4}{k^2}\right)T \\ & \leq 2C'_k + \left(2 - \frac{4}{k^2}\right)\alpha C'_k \end{aligned}$$

Since $\alpha = 1.5$, we obtain a $5 - \frac{6}{k^2}$ approximation algorithm. As before, we can also view this as a constructive proof bounding the ratio of C_k to C'_k. In that case $\alpha = 1$ and we obtain an upper bound of $4 - \frac{4}{k^2}$ on the ratio.

3.1. Odd values of k

When $k > 1$ is odd, we can use our algorithm using $k-1$ as the capacity of the vehicle. Theorem 2.5 and Lemmas 2.6 and 2.7 are true for all values of k, independent of its parity. In Lemmas 2.9 and 2.10, k should be replaced by $k-1$. If the proof is repeated with these modifications, the performance ratio increases by an additive term of $\frac{2}{k-1}$. The performance ratio in this case is at most $5 - \frac{6}{(k-1)^2} + \frac{2}{k-1} \leq 5.1667$.

4. A preemptive algorithm

In this section we describe a simple strategy that also achieves an approximation factor of 5 for the preemptive k-delivery TSP. Our algorithm outputs a preemptive solution (i.e., may drop pegs at intermediate locations during the course of the algorithm), and the length traveled by the vehicle is compared to the length of an optimal preemptive solution.

This algorithm is a modification of the strategy given by Chalasani, Motwani and Rao [10].

4.1. The CMR Algorithm

We first review the algorithm given by Chalasani, Motwani and Rao [10] for this problem. We then show that a modification of the algorithm improves the approximation ratio to 6.5 without preemption, and 5 with preemption. We will assume for simplicity that n is a multiple of k, and that k is even. The former assumption can be made to hold by adding at most $k-1$ dummy peg/slot pairs.

1. Find tours (of almost minimum weight) T_p and T_s of the pegs and slots points respectively. (This step could be implemented using Christofides' heuristic for the TSP.)

2. Break T_p and T_s into paths containing k vertices each, by deleting every k'th edge from each cycle.

3. View each k-path as a "super-node," and construct an auxiliary complete bipartite graph which has one vertex for each of the super-nodes in T_p and T_s. The weight of an edge in this bipartite graph is the shortest distance between a pair of points belonging to the respective super-nodes.

4. Find a minimum-weight perfect matching M in this bipartite graph.

5. Traverse the tour T_p and at the end of each segment use the matching M to transport the k pegs to the corresponding delivery point (super-node) in T_s.

The total length of such a tour is shown to be at most $3T_p + 2T_s + 2M \leq 4.5C_k + 3C_k + \frac{2}{k}A$. As shown in [10], $A \leq kC_k$. Using Christofides' approximation for the TSP, T_p and T_s are at most $1.5C_k$. The approximation ratio obtained is therefore at most 9.5. For geometrical instances, such as points in the plane, the algorithms of Arora [4, 5] and Mitchell [20]) can be used to obtain an $(1+\epsilon)$ approximation of the TSP, and this leads to better approximation factors for these instances.

Theorem 2.5 shows that $A \leq \frac{k}{2}C'_k$. Since $C'_k \leq C_k$, we also have $A \leq \frac{k}{2}C_k$. Using this improved upper bound on the weight of A, the approximation ratio of the algorithm improves from 9.5 to 8.5. A small change to the algorithm improves it further to an approximation factor of 7 as follows. We can traverse T_p in two ways. A clockwise traversal and an anticlockwise traversal gives us two tours whose total length is at most $4T_p + 4T_s + 4M \leq 12C_k + 4(\frac{C_k}{2})$. The smaller of these two tours has length at most the average of these two tour lengths, which is $7C_k$.

Figure 5 illustrates why each segment of the tour T_p is charged at most 4 times by the two tours. Observe that each segment of T_p is traversed once in one of the tours and thrice in the other tour.

4.2. An improved non-preemptive algorithm

We now present an improved algorithm that obtains an approximation ratio of 6.5.

1. Find tours T_p and T_s as before.

2. Break T_p and T_s into paths containing $k/2$ vertices each.

3. View each segment of T_p and T_s as a super-node and construct the auxiliary bipartite graph as before.

4. Find a minimum-weight perfect matching M in this auxiliary graph. Mark the segments of T_p sequentially as $B_1, B_2, \ldots$ around the cycle. Each edge in M matches a segment in T_p to a segment in T_s. Let the segment matched to $B_i, i = 1, 2, \ldots$ be called R_i, and let M_i be the edge of M connecting B_i and M_i.

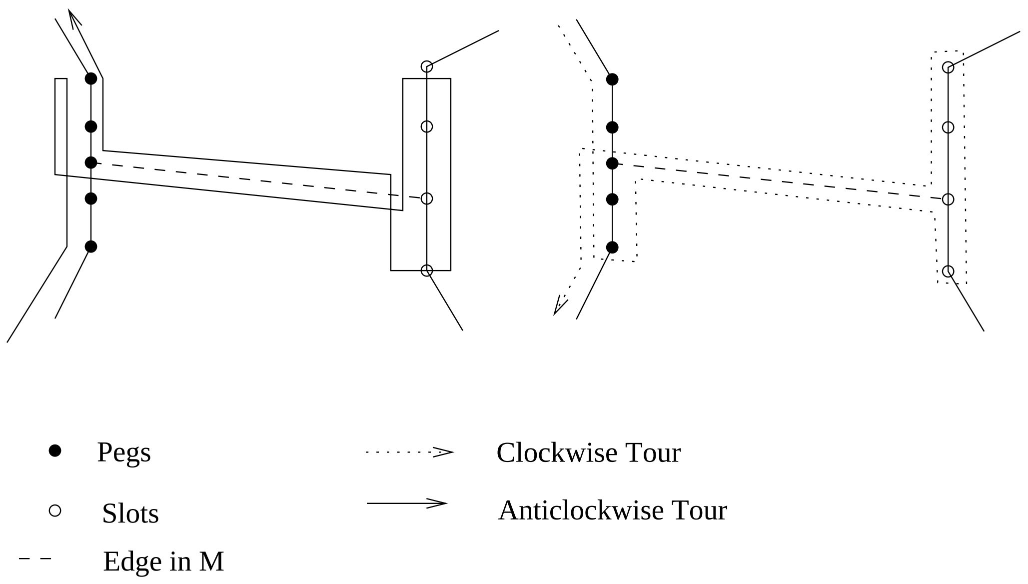

Figure 5: Two tours in T_p.

5. The delivery schedule is as follows. Assume that the vehicle starts at the beginning of segment B_1 with $k/2$ pegs, and proceeds along T_p, picking the pegs in its path. For $i = 1, 2, \ldots$, when it reaches the vertex in B_i incident to the matched edge (M_i) in M, it travels across this edge, and delivers to R_i the pegs that were collected from B_{i-1} (when $i = 1$ we deliver the pegs we started with). After delivering the pegs, it retraces back on M_i and continues along T_p.

6. Finally, when the vehicle returns to the starting location, it is carrying $k/2$ pegs. Lemma 2.3 guarantees that there exists a valid starting point on this traversal, such that the vehicle never run out of pegs or exceeds its carrying capacity.

The algorithm above generates a valid vehicle routing without violating the capacity constraints of the vehicle due to the following reason. When it is on a segment of B_i that precedes M_i on T_p, it is carrying $k/2$ pegs from B_{i-1} and the other pegs that have been collected from B_i. Since B_i has at most $k/2$ pegs, the total number of pegs that it is carrying does not exceed k. On reaching M_i, the vehicle goes to R_i and delivers the $k/2$ pegs that were collected from B_{i-1}. Therefore when it returns to B_i and resumes its journey, it reaches the end of B_i with $k/2$ pegs that were on B_i.

The reason that the algorithm gives a better approximation ratio is as follows. Observe that the algorithm goes around T_p only once (except for segment B_1) instead of twice. This decreases the length traveled. But, the cost of M is now more since there are twice as many segments as before in each of T_s and T_p (because the segments have only $k/2$ vertices each).

Extending the analysis from the previous algorithm, we get $M \leq C'_k$. The vehicle traverses T_p once, T_s twice and M twice. We get the following result: the distance traveled by the vehicle in the scheme devised by the above algorithm is at most $T_p + 2T_s + 2M + \leq 6.5C_k$. Therefore the approximation ratio is at most 6.5.

4.3. An improved preemptive algorithm

In this section, we consider the vehicle routing problem when the vehicle is allowed to drop some pegs at intermediate points in the route, and pick them up later for delivery. We show that the previous algorithm can be modified into a preemptive algorithm with an approximation ratio of a little over 5.

In the previous algorithm, instead of delivering the pegs directly to the slots in R_i, when we cross M_i, we do the following: the vehicle crosses M_i and leaves $k/2$ pegs there to be delivered to the slots later. Once the tour along the peg cycle is completed, the vehicle switches to the slot tour, and delivers the pegs, but this time picking them up at intermediate points on the T_s as it reaches them. In all, the vehicle travels around each of the cycles T_p and T_s once each, twice around M and once extra on the segments B_1 and R_1. By selecting B_1 and R_1 appropriately, over all possible segments, our algorithm obtains a tour with a ratio of $5 + k/n \approx 5$.

Acknowledgment

We are grateful to Greg Frederickson for useful discussions. We thank Shoshana Anily and Julien Bramel for making their paper available to us. We also thank Rafi Hassin for useful comments on an earlier draft of this paper.

References

[1] S. Anily and J. Bramel, "Approximation algorithms for the capacitated traveling salesman problem with pick-ups and deliveries," Manuscript, (1997).

[2] S. Anily and R. Hassin, "The swapping problem," *Networks*, 22:419-433, (1992).

[3] E. Arkin, R. Hassin and L. Klein, "Restricted delivery problems on a network," *Networks*, 29:205-216, (1997).

[4] S. Arora, "Polynomial time approximation schemes for Euclidean TSP and other Geometric problems", *Proc. of 37th Annual Symp. on Foundations of Comp. Sci.* (1996), Pages 1-11.

[5] S. Arora, "Nearly linear time approximation schemes for Euclidean TSP and other Geometric problems", *Proc. of 38th Annual Symp. on Foundations of Comp. Sci.* (1997), Pages 554-563.

[6] T. Asano, N. Katoh, H. Tamaki and T. Tokuyama, "Covering points in the plane by k-tours: towards a polynomial time approximation scheme for general k," *Proc. of 29th Annual ACM Symp. on Theory of Computing (STOC)* (1997), Pages 275-283.

[7] M. J. Atallah and S. R. Kosaraju, "Efficient solutions to some transportation problems with applications to minimizing robot arm travel," *SIAM J. Comput.*, 17(5):569-591, (1988).

[8] J. A. Bondy and U. S. R. Murty, "Graph Theory with Applications," American Elsevier, New York, (1977).

[9] D. O. Casco, B. L. Golden and E. A. Wasil, "Vehicle routing with backhauls: models, algorithms and case studies," *Vehicle Routing: Methods and Studies*, Eds: Golden and Assad, North Holland, Amsterdam (1988).

[10] P. Chalasani, R. Motwani and A. Rao, "Algorithms for robot grasp and delivery," *2nd International Workshop on Algorithmic Foundations of Robotics*, (1996).

[11] N. Christofides, "Vehicle routing," in *The traveling salesman problem*, edited by E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan and D. B. Shmoys, John Wiley & Sons, New York, Pages 431-448, (1985).

[12] G. N. Frederickson, "A note on the complexity of a simple transportation problem," *SIAM J. Comput.*, 22(1):57-61, (1993).

[13] G. N. Frederickson and D. J. Guan, "Non-preemptive ensemble motion planning on a tree," *Journal of Algorithms*,15(1):29-60, (1993).

[14] G. N. Frederickson and D. J. Guan, "Preemptive ensemble motion planning on a tree," *SIAM J. Comput.*, 22(1):1130-1152, (1992).

[15] G. N. Frederickson, M. S. Hecht and C. E. Kim, "Approximation algorithms for some routing problems," *SIAM J. Comput.*, 7(2):178-193, (1978).

[16] M. R. Garey and D. S. Johnson, "Computers and intractability: A guide to the theory of NP-completeness," Freeman, San Francisco, (1978).

[17] M. Haimovich and A. H. G. Rinnooy Kan, "Bounds and heuristics for capacitated routing problems," *Math. Oper. Res.* 10:527-542, (1985).

[18] R. M. Karp, "Two combinatorial problems associated with external sorting," *Combinatorial Algorithms, Courant Comp. Sci. Symp.*, Pages 17-29, Algorithmics Press, New York, (1972).

[19] D. E. Knuth, "The art of computer programming, vol 3: Sorting and Searching," Addison Wesley, (1973).

[20] J. Mitchell, "Guillotine subdivisions approximate polygonal subdivisions: Part II – A simple polynomial-time approximation scheme for geometric k-MST, TSP, and related problems," Manuscript, (1996).