A Quasi-Polynomial-Time Approximation Scheme for Vehicle Routing on Planar and Bounded-Genus Graphs*

Amariah Becker[1], Philip N. Klein[2], and David Saulpic[3]

1 **Department of Computer Science, Brown University, Providence, RI, USA**
`becker@cs.brown.edu`
2 **Department of Computer Science, Brown University, Providence, RI, USA**
`klein@cs.brown.edu`
3 **Department of Computer Science, École Normale Supérieure, Paris, France**
`david.saulpic@ens.fr`

Abstract

The Capacitated Vehicle Routing problem is a generalization of the Traveling Salesman problem in which a set of clients must be visited by a collection of capacitated tours. Each tour can visit at most Q clients and must start and end at a specified depot. We present the first approximation scheme for Capacitated Vehicle Routing for non-Euclidean metrics. Specifically we give a quasi-polynomial-time approximation scheme for Capacitated Vehicle Routing with fixed capacities on planar graphs. We also show how this result can be extended to bounded-genus graphs and polylogarithmic capacities, as well as to variations of the problem that include multiple depots and charging penalties for unvisited clients.

1998 ACM Subject Classification G.2.2 Graph Algorithms

Keywords and phrases Capacitated Vehicle Routing, Approximation Algorithms, Planar Graphs

Digital Object Identifier 10.4230/LIPIcs.ESA.2017.12

1 Introduction

Vehicle routing refers to a class of problems in which one selects routes for a vehicle that must make deliveries or pickups at specified locations. Irnich et al., in the introductory chapter [14, p. 3] of a book on vehicle routing, define the capacitated vehicle routing problem (CVRP) as follows: there is a (directed or undirected) graph G with edge costs, a distinguished vertex r called the *depot*, and for each vertex v, a demand $q(v)$. Finally, there is a number Q, which is the capacity of the vehicle(s). A solution is a set of *tours*, where each tour starts and ends at the depot and serves the demands of some of the vertices it visits. Each tour must serve a total demand of at most Q, and every demand must be served by one of the tours. The goal is to find a solution whose total cost is minimum.

To be consistent with the algorithms literature, we use a slightly different definition of capacitated vehicle routing: we assume that the demands are all 0 or 1, i.e. that there is a set Z of vertices, called the *clients*, where the demand is 1, and demands at other vertices are zero. (One can model multiple clients located at the same vertex v by introducing artificial vertices, adjacent to v via artificial edges of cost zero.)

* This research was supported by National Science Foundation grant CCF-1409520.

The problem is APX-hard for an arbitrary graph when $Q \geq 3$ [5], and to approximate it within a factor 1.5 is NP-complete even in a tree when Q is unbounded [10]. We are interested in finding solutions that are within a factor $1+\epsilon$ of optimal for any given ϵ. However, despite the fact that the problem is often described as a problem in **road networks**, theoretical work on algorithms achieving $1+\epsilon$ approximation has been restricted to the Euclidean case.

Since the family of planar graphs (or more generally graphs of bounded genus) are useful for modeling road networks,[1] it is desirable to find an algorithm that achieves a $1+\epsilon$ approximation on such graphs with arbitrary nonnegative edge costs. Before this work, no such approximation scheme was known for any graph class (except trees, where the problem is polynomial-time-solvable for fixed capacity).

▶ **Theorem 1.** *For any $\epsilon > 0$ and any $Q > 0$, there is a quasi-polynomial-time algorithm that, given an instance of* CAPACITATED VEHICLE ROUTING *in which the capacity is Q and the graph is planar, finds a solution whose cost is at most $1+\epsilon$ times optimum.*

A family of algorithms of this form, where for each $\epsilon > 0$ there is an algorithm that achieves a $1+\epsilon$ approximation, is an *approximation scheme*. By *quasi-polynomial* is meant a function $f(n)$ that is $O(n^{\log^c n})$ for some constant c.

As pointed out in [14], with a limited fleet, it may be impossible to service all requests, and there is an advantage in simultaneously optimizing both routing and request selection. We model this using a natural generalization of the capacitated-vehicle-routing problem: an instance specifies also a *penalty* for each client; the solution is allowed to miss some clients and the goal is to find a solution that minimizes the sum of cost plus penalties. We call this CAPACITATED VEHICLE ROUTING WITH PENALTIES.

This generalization can handle the *vehicle routing problem with private fleet and common carrier* (VRPPC), “where customers may either be served by using owned vehicles with traditional routes or be assigned to a common carrier, which serves them directly at a prefixed cost” [14, p. 13].

Our quasi-polynomial-time approximation scheme can also be extended to handle CAPACITATED MULTIPLE-DEPOT VEHICLE ROUTING. In this version, several vertices are designated as depots, and tours can start and end at different depots.

The algorithm can also handle a graph of bounded genus and a capacity Q that is polylogarithmic. (Q is considered constant in Theorem 1.)

▶ **Theorem 2.** *For any $\epsilon > 0$, any $g \geq 0$, any $R \geq 0$ and any $c \geq 0$, there is a quasi-polynomial-time algorithm that, given an instance of* CAPACITATED MULTIPLE-DEPOT VEHICLE ROUTING WITH PENALTIES *in which the capacity Q is $O(\log^c n)$ and the graph has genus g with R designated depots, finds a solution whose cost is at most $1+\epsilon$ times optimum.*

1.1 Related work

CAPACITATED VEHICLE ROUTING is a generalization of TRAVELING SALEMAN PROBLEM (for TSP $Q = n$).

Haimovich and Rinnoy Kan [12] showed the following:

▶ **Lemma 3.** *For* CAPACITATED VEHICLE ROUTING *with capacity Q and client set Z,*

$$OPT \geq \frac{2}{Q} \sum \{d(c,r) \ : \ c \in Z\}. \tag{1}$$

[1] Aside from highways, the nonplanarities in road networks tend to be localized, and informally approximation schemes for planar graphs can often “work around” these localized nonplanarities.

This lemma implies a constant-factor approximation in general metrics, where the constant depends on the approximation ratio for TSP. Capacitated Vehicle Routing in general graphs is APX-hard for every fixed $Q \geq 3$ [4, 5]. Haimovich and Rinnoy Kan [12] gave a polynomial-time approximation scheme (PTAS) for the Euclidean plane for the case when the capacity Q is constant. Asano et al. [5] gave an algorithm that was a PTAS when Q is $O(\log n/\log\log n)$. Mathieu and Das [7] gave a quasi-polynomial-time approximation scheme that handles arbitrary Q. Building on [7], Adamaszek, Czumaj, and Lingas [1] give a polynomial-time approximation scheme that for any $\epsilon > 0$ can handle Q up to $2^{\log^\delta n}$ where δ is a positive number that depends on ϵ. There has been some work on approximation schemes to $\mathbb{R}^3$ [16] and $\mathbb{R}^d$ [15] but these require that Q be $O(\log^{1/d}\log n)$. No polynomial-time approximation scheme is known for arbitrary Q, even for $\mathbb{R}^2$.

There is little known about approximation of vehicle routing in special metrics other than low-dimensional Euclidean metrics. Hamaguchi and Katoh [13] and Asano, Katoh, and Kawashima [3] gave better constant-factor approximation algorithms for the case where the graph is a tree.

1.2 Our Approach

Our algorithm uses a recursive decomposition of the graph via shortest-path separators. That is, there is a recursive clustering in which the vertices on the boundary of each cluster lie on a small number of shortest paths. This general idea has been used in several previous approximation schemes for planar and bounded-genus graphs [2, 6, 8, 11]. The closest previous use was in addressing the k-center problem [8], and we use a lemma from that paper stating that such a recursive decomposition exists that has logarithmic depth.[2]

This paper introduces several new ideas in order to apply the recursive decomposition to vehicle routing. Before finding the recursive decomposition, the algorithm must *prune* the graph to eliminate vertices too far from the depot to participate in an optimal solution. The shortest paths bounding each cluster are *subpaths of shortest paths from the depot.* This ensures that if one vertex of a bounding shortest path is farther along the bounding shortest path than another, it is also farther from the depot. This in turn is useful since, as we saw in Section 1.1, there is a lower bound (3) on OPT that is based on the distance of clients from the depot.

Some of the vertices of these bounding shortest paths are designated as *portals*, and (some) paths of the solution are restricted to entering and leaving clusters via portals. This in itself is not novel; portals have been used before. We introduce two new ideas. In previous use of portals in approximation schemes for planar graphs, portals are selected uniformly along a path. In this paper, it is essential that the portals be selected in a nonuniform fashion: the farther from the depot, the greater the spacing between portals. This introduces more error in areas of the graph farther from the depot but such error can be tolerated due to the lower bound (3).

Second, requiring the entire solution to pass in and out of clusters via portals would introduce too much error. Instead, we only require a tour to use portals in between picking up clients. That way, we can bound the error in terms of clients and their distance from the depot. (This error also depends on the depth of nesting of the recursion, since in the process

[2] Such a decomposition was earlier described by Thorup [17] in the context of approximate distance oracles. However, in addressing the generalization to multiple depots we use a generalization of the decomposition that follows from slight modification of the proof in [8].

of picking up a client the tour might pass through many clusters at different levels of nesting, but the depth of nesting is merely logarithmic.)

Having shown that an (approximately) optimal solution can be assumed to use clusters in this way, we reduce the problem to one we can address using dynamic programming. In most previous work on approximation schemes for planar graphs, this consists in simply finding an optimal solution in a graph of bounded branchwidth (or treewidth) but because the solution can pass through the boundary without using portals, that does not quite suffice in our case; the dynamic program must also make use of the metric on the entire graph as well.[3]

Paper Outline. Section 2 provides preliminary definitions. In Section 3 we describe the graph decomposition and portal selection. In Section 4 we prove a structure theorem that shows a near-optimal solution with the restricted structure exists. Section 5 provides the dynamic program that finds such a near optimal solution in quasi-polynomial time. Finally in Section 6 we describe several generalizations of our result.

2 Preliminaries

Let $G = (V, E)$ be a graph. We denote the vertex set of G by $V(G)$. G is *planar* if it can be embedded on the surface of a sphere without any edge crossings. We let $n = |V|$. A planar graph with n vertices and no parallel edges has $O(n)$ edges.

For any edge set $F \subseteq E$ the *boundary* of F, denoted $\partial(F)$, is the set of vertices that are incident both to edges in F and edges in $E - F$.

We use $d(u, v)$ to denote the (shortest path) distance from u to v. We can easily compute all-pairs shortest paths in polynomial time, so we assume throughout the paper that we have access to all (precomputed) distances. Additionally we assume that the cost of any edge (u, v) is $d(u, v)$; if not, it would not be used and can be removed from the graph. We use $d(P) = \sum_{(u,v)\in P} d(u, v)$ to denote the cost of a path P.

For a graph G and vertex r, an *r-rooted shortest path tree* T is an r-rooted tree in which for all v in V the r-to-v path in T is a shortest path. For any vertex u on a shortest r-to-v path, we call the u-to-v subpath a *from-r shortest subpath.* Such a subpath must also be a shortest u-to-v path.

A *triangulated* planar graph is one in which every face has exactly three incident edges. A planar graph can easily be triangulated in linear time by adding edges that recursively subdivide faces with more than three incident edges. Each new edge (u, v) is given cost $d(u, v)$. Triangulating a planar graph requires adding $O(n)$ edges.

A *recursive partition* of a set Y is a rooted binary tree in which each node is labeled with a *cluster* $\mathcal{C} \subseteq Y$ such that the root node is labeled with $\mathcal{C} = Y$, and for any node labeled with cluster $\mathcal{C}_0$ the children nodes are labeled with clusters $\mathcal{C}_1$ and $\mathcal{C}_2$ that form a partition of $\mathcal{C}_0$ [8].

A *recursive clustering* of a graph G is a rooted binary tree in which

- each node is labeled with a *cluster* $\mathcal{C} \subseteq V(G)$,
- If x is a child of y then the cluster associated with x is a subset of the cluster associated with y, and
- there is a mapping $\phi(\cdot)$ that maps each vertex v of G to a leaf cluster $\phi(v)$ that contains v.

Each vertex v is considered to be *assigned to* each cluster containing $\phi(v)$.

[3] A similar technique was used in [8].

A vertex v of a cluster $\mathcal{C}$ is a *boundary vertex* of the cluster if v also belongs to a cluster $\mathcal{C}'$ that neither contains nor is contained in C. An edge uv of G is a *boundary edge* of the cluster if u is in the cluster and v is not. The *depth* of a recursive clustering is the depth of the rooted binary tree.

For an instance of Capacitated Vehicle Routing, Z is the set of clients, r is the depot, and Q is the capacity. A solution is a collection of *tours*, each starting and ending at r and visiting at most Q clients.

3 Decomposing the Graph

3.1 Graph Pruning

As a preprocessing step, the algorithm prunes from the graph those vertices that have no clients and are very far from the depot. Specifically, the algorithm deletes each vertex that does not lie on some u-to-v shortest path with $u, v \in Z \cup \{r\}$. Since the optimal solution is composed of such paths, pruning does not increase OPT.

▶ **Lemma 4.** *For all vertices w that remain after the preprocessing step, $d(r, w) \leq OPT$*

Proof. Since w survived the pruning step, it must lie on some u-to-v shortest path with $u, v \in Z \cup \{r\}$. Without loss of generality, assume that the optimal solution visits u before visiting v. Therefore $OPT \geq d(r, u) + d(u, v) \geq d(r, u) + d(u, w) \geq d(r, w)$ where the final inequality comes from the triangle inequality. ◀

3.2 Cluster Decomposition

The following lemma is a slight generalization of a lemma in [8] (though it is probably folklore):

▶ **Lemma 5** (Generalization of Lemma 3.1 in [8])**.** *Let T be a tree of degree at most three. Let Y be a subset of the vertices. There is a depth-$O(\log |Y|)$ recursive clustering of T such that*

- *there are no boundary vertices,*
- *each leaf cluster is assigned only one vertex of Y, and*
- *each cluster C has at most four boundary edges.*

Let G be a planar embedded graph and let T be a spanning tree of G. Let G' be obtained from G by adding artificial edges to triangulate G. Let G^* be the planar dual of G'. Let T^* be the set of edges of G^* corresponding to edges of G' that are not in T. Then T^* is a spanning tree of G^* (the *interdigitating tree*). Each edge of T^* corresponds to a nontree edge in G, which in turn defines a cycle consisting of that nontree edge together with a path through T. Since G' is triangulated, G^* has degree at most three. Map each vertex of G' to one of the faces to which it is incident. Let Y be the faces to which elements of Z are mapped. By choosing T to be an r-rooted shortest-path tree of G and applying Lemma 5, we obtain the following generalization of a corollary of [8].

▶ **Lemma 6** (Generalization of Corollary 3.1 of [8])**.** *Let G be a planar embedded triangulated graph with edge costs, let Z be a subset of the vertices, and let r be a vertex of G. There is a depth-$O(\log |Z|)$ recursive clustering of G with the following properties:*

- *there are no boundary edges,*
- *for each cluster, there are at most eight from-r shortest paths such that the boundary vertices of the cluster are the vertices that lie on these paths, and*
- *at most three vertices of Z are assigned to each leaf cluster.*

The algorithm computes a recursive clustering as described in Lemma 6.

3.3 Choosing Portals

The algorithm designates *portals* along each cluster boundary in a two-step process. First it designates some of the vertices as *spacers*. Second, for each cluster it designates as portals a subset of the cluster's boundary vertices, including those boundary vertices that are spacers and also some additional vertices.

The choice of spacers depends on a parameter δ. We will choose

$$\delta = \frac{\epsilon}{(Q+4)c\log|Z|} \tag{2}$$

where c is an absolute constant to be determined in the proof of Theorem 11.

We first describe spacer selection. Let T be the shortest-path tree. Let $\hat{v}$ be the vertex farthest from r that remains after the pruning step and let $\delta > 0$. Consider the unpruned vertices in increasing order of distance from r. For each vertex v in turn, designate v as a spacer if no ancestor in T of v within distance $\delta\ \max(d(r, s_{i-1}), \frac{1}{|Z|}d(r,\hat{v}))$ has been designated a spacer.

▶ **Lemma 7.** *Each from-r shortest path has at most $2 + \log_{1+\delta}\frac{|Z|}{\delta}$ spacers.*

Proof. Let P be a from-r shortest path, and let $r{=}s_0, s_1, \ldots, s_\ell$ be the spacers on P in increasing order of distance from r. We bound ℓ as follows. For each $i > 0$, $d(s_{i-1}, s_i) > \delta\ d(r, s_{i-1})$, so

$$d(r, s_i) = d(r, s_{i-1}) + d(s_{i-1}, s_i) > (1+\delta)d(r, s_{i-1})\,.$$

By induction,

$$d(r, s_\ell) > (1+\delta)^{\ell-1} d(r, s_1)\,.$$

Since $d(r, s_1) > \delta\frac{1}{|Z|}d(r,\hat{v})$, we infer $d(r, s_\ell) > (1+\delta)^{\ell-1}\frac{\delta}{|Z|}d(r,\hat{v})$, which implies

$$(1+\delta)^{\ell-1} < \frac{|Z|}{\delta}\frac{d(r,s_\ell)}{d(r,\hat{v})} \le \frac{|Z|}{\delta}$$

which shows

$$\ell - 1 < \log_{1+\delta}\frac{|Z|}{\delta}.$$ ◀

The algorithm then designates some of each cluster's boundary vertices as *portals*. For each cluster $\mathcal{C}$, the boundary vertices of $\mathcal{C}$ lie on $O(1)$ from-r shortest subpaths. For each such from-r subpath P, designate as portals the first vertex of P and all the vertices of P that are spacers.

By Lemma 7, each path P contributes $O(\delta^{-1}\log(\delta^{-1}|Z|))$ portals. Using the definition of δ in Equation 2, we obtain

▶ **Lemma 8.** *For each cluster boundary, the above algorithm designates $O(Q\epsilon^{-1}\log^2|Z|)$ portals.*

We also show a bound on the distance along the boundary to the nearest portal.

▶ **Lemma 9.** *For every cluster $\mathcal{C}$ and boundary vertex $v \in \partial(\mathcal{C})$, there is a portal p of $\mathcal{C}$ and a p-to-v path of cost at most $\delta\ (d(r,v) + OPT/|Z|)$.*

Proof. By Lemma 6, v must lie on some from-r shortest subpath of a from-r shortest path P. Let s be v's closest ancestor in T that is a spacer. By the algorithm for designating spacers,

$$d(s,v) \leq \delta \ \max(d(r,s), \frac{1}{|Z|}d(r,\hat{v})) \leq \delta \ \max(d(r,v), OPT/|Z|)\,.$$

If s belongs to P then s is a boundary vertex of $\mathcal{C}$ and hence a portal of $\mathcal{C}$. In this case, we take $p = s$. If not, then let p be the first vertex of P and note that $d(p,v) \leq d(s,v)$. ◀

4 Structure Theorem

Consider a solution to Capacitated Vehicle Routing. It consists of a set of tours. Associated with each tour P is a set of vertices in $V(P) \cap Z$ that the tour is considered to visit. For a cluster $\mathcal{C}$, a tour *fragment* with respect to $\mathcal{C}$ is a maximal subpath of a tour all of whose vertices are in C. Every tour starts and ends at the depot r. If r is in $\mathcal{C}$ then it is a boundary vertex of $\mathcal{C}$. Therefore, by maximality, each endpoint of a tour fragment with respect to $\mathcal{C}$ is a boundary vertex of $\mathcal{C}$.

A tour fragment is *visiting* if it visits clients and *passing* if it does not. The endpoints of a visiting fragment are called *gates.*

▶ **Lemma 10.** *Any solution to* Capacitated Vehicle Routing *crosses through* $O(\log|Z|)$ *gates between two consecutive visits to clients.*

Proof. Consider a solution to Capacitated Vehicle Routing. Let u and v be two consecutive clients visited by the solution, and let P be the subpath of the tour between its visit to u and its visit to v. Let C be a cluster and let S be a visiting tour segment with an endpoint x on P. Since S is a visiting segment, it visits some vertex. Since no visits occur on P between u and v, the other endpoint y of P must not be an internal vertex of P. Thus either u or v must be assigned to the cluster C. If u is assigned to C then all edges on the u-to-x subpath of P belong to C, and the edge of P after x does not. If v is assigned to C then all edges on the x-to-v subpath of P belong to C, and the edge of P before x does not.

Let $\mathcal{C}_{u,0} \subset \mathcal{C}_{u,1} \subset \ldots \subset \mathcal{C}_{u,\ell}$ be the clusters that are assigned u, and let $\mathcal{C}_{v,0} \subset \mathcal{C}_{v,1} \subset \ldots \subset \mathcal{C}_{v,k}$ be the clusters that are assigned v.

For $i = 0, 1, \ldots, \ell$, let $t_{u,i}$ be the last vertex x of P such that the u-to-x subpath of P consists of edges of cluster $C_{u,i}$. For $i = 0, 1, \ldots, k$, let $t_{v,i}$ be the first vertex x of P such that the x-to-u subpath of P consists of edges of cluster $C_{v,i}$.

Since the clusters are non-crossing, P visits $t_{u,0}, t_{u,1}, \ldots, t_{u,\ell}, t_{v,k}, t_{v,k-1}, \ldots, t_{v,0}$ in order (See Figure 1a). The argument above shows that every gate is one of $t_{u,0}, \ldots, t_{v,0}$.

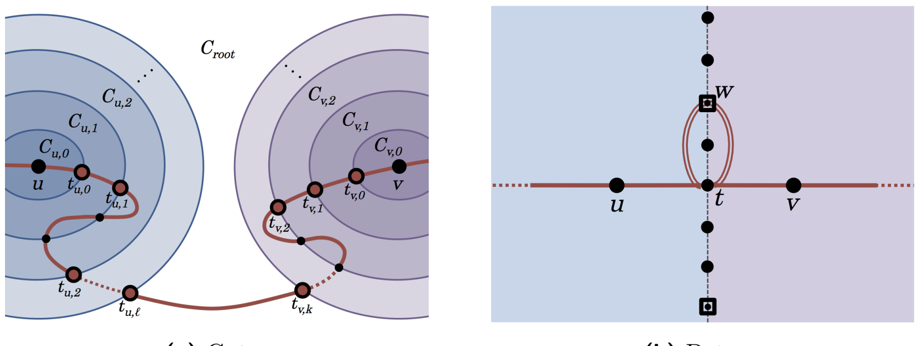
By Lemma 6, the depth of the decomposition is $O(\log|Z|)$, so $k + \ell$ is $O(\log|Z|)$. ◀

Now we break up tours in a different way. Again consider a solution to Capacitated Vehicle Routing. It consists of a set of tours. For a cluster C, a *tour segment* P with respect to $\mathcal{C}$ is *portal-respecting* if P has a portal-to-portal subpath P' such that every vertex not in P' is a boundary vertex of $\mathcal{C}$.

▶ **Theorem 11.** *There exists a portal-respecting solution with cost at most* $(1+\epsilon)OPT$.

Proof. Let $\mathcal{S}^*$ be an optimal solution to Capacitated Vehicle Routing. To prove the theorem, we show how to modify $\mathcal{S}^*$ to construct a portal-respecting solution $\hat{\mathcal{S}}$ by introducing *detours* at every gate, and bound the cost incurred by these detours.

Consider a tour fragment P with respect to some cluster $\mathcal{C}$, and let t be an endpoint of P. The corresponding detour is the subpath of a from-r shortest subpath from t to the



(a) Gates **(b)** Detour

Figure 1 (a) Gates are depicted by large, hollow circles at crossings. Non-gate crossings enclose passing segments (b) Boundary portals are denoted by squares. The double-line paths depict a detour.

nearest portal, and back. (See Figure 1b.) Splicing such a detour into the solution at each gate ensures that the solution is still feasible and is portal-respecting. It remains to show that the cost of these detours is small.

Let u and v be two consecutive clients visited by $\mathcal{S}^*$. By Lemma 10 there is some constant c such that there are at most $c \log |Z|$ gates between u and v where detours will be added. We use this constant c in the definition of δ given in Equation 2. By Lemma 9 the distance from any crossing t to the nearest portal is at most $\delta\ (d(r,t) + OPT/|Z|)$, so the cost of each detour is at most $2\delta\ (d(r,t) + OPT/|Z|)$.

Since $\mathcal{S}^*$ is optimal, the path that $\mathcal{S}^*$ takes from u to v must be a shortest path. By the triangle inequality, $d(r,t) \leq d(r,v) + d(v,t) \leq d(r,v) + d(v,u)$. Therefore, the cost of each detour is at most $2\delta\ (d(u,v) + d(r,v) + OPT/|Z|)$. Summing over all gates gives $2\delta c \log |Z| \big(d(u,v) + d(r,v) + OPT/|Z|\big)$, and summing over all pairs of consecutive clients and using Lemma 3,

$$\sum_{(u,v)\in\mathcal{S}^*} 2\delta c \log |Z| \big(d(u,v) + d(r,v) + OPT/|Z|\big) \leq 2\delta c \log n (OPT + \frac{Q}{2} OPT + OPT)$$

$$= \delta c \log |Z| (Q+4) OPT\,.$$

The definition of δ given in Equation 2 ensures that the total detour cost is at most ϵOPT. ◀

The notion of *portal-respecting* can be applied not just to a solution but to a partial solution as well. This is used in the next section.

5 Dynamic Program

We present two dynamic programs: the first is slow but is the basis of the second, which gives a QPTAS. Both have the same configurations but enumerate the transitions in different ways.

5.1 Configurations

For each cluster, the dynamic program computes the minimal cost of a *portal-respecting* solution that visits all the clients assigned to the cluster. A *configuration* for a cluster $\mathcal{C}$

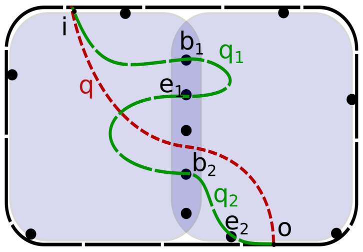


Figure 2 The segment (i, o, q) of the parent cluster is shown in red (short-dashed line). The green path (long-dashed line) shows one way to map this segment onto the child clusters: segment (b_1, e_1) visits q_1 clients and (b_2, e_2) visits q_2 clients with $q_1 + q_2 = q$. Segments (i, b_1), (e_1, b_2), and (e_2, o) are passing segments and visit no clients

describes the tour segments formed by the intersection of $\mathcal{C}$ with a solution. Recall that if the depot r is contained in a cluster, it is a portal of its boundary. Additionally, since each client is assigned to exactly one leaf cluster, we avoid overcounting any client's demand. For a client z, let $D_{\mathcal{C},z}$ be the demand of z inside the cluster $\mathcal{C}$. $D_{\mathcal{C},z} = 1$ if z is assigned to $\mathcal{C}$, otherwise $D_{\mathcal{C},z} = 0$.

A *configuration* $\mathcal{X}$ for cluster $\mathcal{C}$ specifies, for each pair of portals (i, o) of the cluster and for each $q \in \{1, ..., Q\}$, a number $\mathcal{X}_{i,o,q}$ of segments that enter $\mathcal{C}$ at portal i, visit exactly q clients in $\mathcal{C}$, and leave $\mathcal{C}$ at portal o. A configuration for cluster $\mathcal{C}$ is *admissible* if $\Sigma_{i,o,q} q\mathcal{X}_{i,o,q} = \Sigma_{z\in\mathcal{C}} D_{\mathcal{C},z}$.

A *partial solution* S for cluster $\mathcal{C}$ is a set of tour segments that stays inside the cluster. We say that a partial solution S for cluster $\mathcal{C}$ *induces the configuration* $\mathcal{X}$ if every *visiting* segment of S corresponds to a segment described in $\mathcal{X}$ (recall that a visiting segment is one that visits a client).

Our dynamic program computes, for each cluster $\mathcal{C}$ and admissible configuration $\mathcal{X}$, the weight $DP(\mathcal{C}, \mathcal{X})$ of the minimum-weight, portal-respecting partial solution that induces the configuration $\mathcal{X}$.

5.2 Compatibility

To compute the minimal cost of a partial solution for a cluster $\mathcal{C}_0$ that induces the configuration $\mathcal{X}^0$, the algorithm determines possible configurations for the two child clusters of $\mathcal{C}_0$, namely $\mathcal{C}_1$ and $\mathcal{C}_2$. Let $\mathcal{C}_0$ be a cluster, with two child clusters $\mathcal{C}_1$ and $\mathcal{C}_2$, and let $\mathcal{X}^0$, $\mathcal{X}^1$, and $\mathcal{X}^2$ be three configurations such that $\mathcal{X}^\imath$ is admissible for $\mathcal{C}_\imath$. We say that $\mathcal{X}^1$ and $\mathcal{X}^2$ are *compatible* with $\mathcal{X}^0$ if each segment (i, o, q) described in $\mathcal{X}^0$ can be mapped to a tuple of segments of $\mathcal{X}^1$ and $\mathcal{X}^2$, $\big((b_1, e_1, j_1, q_1), \cdots, (b_K, e_K, j_K, q_K)\big)$, where $j_\imath \in \{1, 2\}$, $b_\imath$ and $e_\imath$ are portals of $\mathcal{C}_{j_\imath}$, and such that $0 < q_\imath < Q$ and $\Sigma_{\imath=1}^{K} q_\imath = q$. We use $(\mathcal{X}^1, \mathcal{X}^2) \sim \mathcal{X}^0$ to denote that $\mathcal{X}^1$ and $\mathcal{X}^2$ are compatible with $\mathcal{X}^0$. To ensure that partial solutions are portal-respecting, configurations only need to describe the segments that visit clients. The other segments need not cross boundaries at portals, so we assume them to be shortest paths in the original graph.

Conversely, every segment of $\mathcal{X}^1$ and $\mathcal{X}^2$ must correspond to exactly one segment of $\mathcal{X}^0$. Intuitively, this means that every segment in $\mathcal{X}^0$ can be broken into subsegments that respect the portals of $\mathcal{C}_1$ and $\mathcal{C}_2$. The path from i to o can be divided into a shortest path from i to b_1, segments from $b_\imath$ to $e_\imath$ visiting $q_\imath$ clients in the cluster $\mathcal{C}_{j_\imath}$, and shortest paths from $e_\imath$ to $b_{\imath+1}$ and from e_K to o visiting 0 clients.

We define the *price*, P, of the compatible configurations to be the cost of connecting the segments: $P(\mathcal{C}_0, \mathcal{X}^0, \mathcal{X}^1, \mathcal{X}^2) = d(i, b_1) + \Sigma_{\imath=1}^{K-1} d(e_\imath, b_{\imath+1}) + d(e_K, o)$. Therefore, the minimal cost of a partial solution for a cluster $\mathcal{C}_0$ that induces the configuration $\mathcal{X}^0$ is $DP(\mathcal{C}_0, \mathcal{X}^0) = \min_{(\mathcal{X}^1,\mathcal{X}^2)\sim\mathcal{X}^0} DP(\mathcal{C}_1, \mathcal{X}^1) + DP(\mathcal{C}_2, \mathcal{X}^2) + P(\mathcal{C}_0, \mathcal{X}^0, \mathcal{X}^1, \mathcal{X}^2)$.

The algorithm enumerates all possible configurations $\mathcal{X}^1$ and $\mathcal{X}^2$ that are compatible with $\mathcal{X}^0$. As in the above definitions, the algorithm breaks every segment of $\mathcal{X}^0$ into subsegments, each visiting some clients in $\mathcal{C}_1$ or in $\mathcal{C}_2$. It then adds each subsegment to the corresponding subconfiguration: the subsegment is added to $\mathcal{X}^{j_\imath}$. As $q_\imath > 0$ and $\Sigma_{i=\imath}^{K} q_\imath = q \leq Q$, $K \leq Q$. The algorithm enumerates all possibilities and calculates the value of $DP(\mathcal{C}_0, \mathcal{X}^0)$.

5.3 Base Cases

Each base case is a cluster in which there are at most three clients. It is therefore straightforward to find the minimal cost of a configuration.

5.4 Final Output and correctness

Since the topmost cluster has r as its only portal, all configurations will consist of r-to-r segments visiting at most Q clients, and collectively visiting all clients of Z. These are exactly the feasible Capacitated Vehicle Routing solutions. The algorithm returns the minimum over all admissible configurations of the top-level cluster, which is the cost of the optimal portal-respecting solution.

▶ **Theorem 12.** *The dynamic programming algorithm described above outputs the minimal weight of a portal-respecting solution to* Capacitated Vehicle Routing.

The proof is omitted here due to space limitations.

5.5 Complexity Analysis

The complexity is determined by the number of compatible configurations. For each cluster and each admissible configuration for this cluster, the algorithm computes

$$|\{\text{ways of breaking a segment}\}|^{|\{\text{segments}\}|}$$

compatible subconfigurations. We first count the number of admissible configurations for a cluster.

▶ **Lemma 13.** *The number of admissible configurations $\mathcal{X}$ for a given cluster $\mathcal{C}$ is*

$$|Z|^{O(Q^3\epsilon^{-2}\log^2|Z|)}.$$

Proof. An admissible configuration is a vector $\mathcal{X}$ indexed by two portals (i, o) and a number of clients q. $\mathcal{X}_{i,o,q}$ is the number of segments going from i to o visiting q clients. As the configuration is admissible, $\Sigma_{i,o,q} q\mathcal{X}_{i,o,q} = \Sigma_{z\in\mathcal{C}} D_{\mathcal{C},z} \leq |Z|$ (because $D_{\mathcal{C},z} \in \{0, 1\}$). Therefore $\mathcal{X}_{i,o,q} \leq |Z|$. Moreover, $q \leq Q$ and because by Lemma 8 there are $O(Q\epsilon^{-1}\log|Z|)$ portals for $\mathcal{C}$ the number of choices of (i, o) is less than $O(Q^2\epsilon^{-2}\log^2|Z|)$. The number of admissible configurations $\mathcal{X}$ for a given cluster is thus $|Z|^{O(Q^3\epsilon^{-2}\log^2|Z|)}$ ◀

We now count the number of compatible subconfigurations for a given configuration $\mathcal{X}$.

▶ **Lemma 14.** *There are $O\big((2Q^3\epsilon^{-2}\log^2|Z|)^{Q|Z|}\big)$ compatible subconfiguration pairs for a given configuration.*

Proof. Using the same argument as in Lemma 13, $\Sigma_{i,o,q} q\mathcal{X}_{i,o,q} \leq |Z|$, and as $q > 0$ we can bound the number of segments of a configuration: $\Sigma_{i,o,q}\mathcal{X}_{i,o,q} \leq |Z|$. To break a segment, the algorithm chooses at most Q subsegments, each one consisting of a boolean, a pair of portals and a number of clients visited by the segment (see Section 5.2). As there are $O(Q\epsilon^{-1}\log|Z|)$ child-cluster portals by Lemma 8 and the capacity is bounded by Q, there are $O((2 \cdot Q \cdot Q^2\epsilon^{-2}\log^2|Z|)^Q)$ ways of breaking a single segment. As there are fewer than $|Z|$ segments, we have $O((2Q^3\epsilon^{-2}\log^2|Z|)^{Q|Z|})$ compatible subconfiguration pairs per configuration. ◀

▶ **Lemma 15.** *The overall complexity of the dynamic program is*

$$|Z|^{O(\frac{Q^3\log^4|Z|}{\epsilon^2})} \cdot O(\frac{Q^3\log^4|Z|}{\epsilon^2})^{Q|Z|}.$$

Proof. As stated above, the dynamic program computes for each cluster and each admissible configuration, all compatible subconfigurations. As the decomposition is a binary tree with at most one leaf per client, the number of clusters is $O(Z)$. Combining this with Lemma 13 and Lemma 14, the total complexity is therefore $O\big(|Z| \cdot |Z|^{O(Q^3\epsilon^{-2}\log^2|Z|)}\big(2Q^3\epsilon^{-2}\log^2|Z|\big)^{Q|Z|}\big)$. ◀

5.6 QPTAS

The slowest operation in the DP is generating all compatible subconfigurations for a given cluster. But this is very redundant: two different segments can be broken into the same subsegments. We present a preprocessing step that computes, for every cluster and every triplet of configurations for parent and children clusters, the minimal price $P(\mathcal{C}_0, \mathcal{X}^0, \mathcal{X}^1, \mathcal{X}^2)$ of *passing* segments needed to make the configurations compatible. Recall that a passing segment is one that visits no clients and that they are necessary to connect the *visiting* segments of the subconfigurations.

5.6.1 Preprocessing Algorithm

We describe a recursive algorithm. The entries are a cluster $\mathcal{C}_0$, a configuration $\mathcal{X}^0$ of $\mathcal{C}_0$ and two configurations $\mathcal{X}^1$ and $\mathcal{X}^2$ of the children of $\mathcal{C}_0$. To determine the price for connecting these three configurations, the algorithm considers the first segment appearing in $\mathcal{X}^0$ and breaks it into subsegments belonging to the children. It then deletes this segment from $\mathcal{X}^0$ (giving a new configuration $\hat{\mathcal{X}}^0$) and deletes the subsegments from the children configurations, to obtain the subconfigurations $\hat{\mathcal{X}}^1$ and $\hat{\mathcal{X}}^2$. It tries all possibilities for breaking this segment and returns the cheapest one :

$$P[\mathcal{C}_0, \mathcal{X}^0, \mathcal{X}^1, \mathcal{X}^2] = \min\big(P(\mathcal{C}_0, \hat{\mathcal{X}}^0, \hat{\mathcal{X}}^1, \hat{\mathcal{X}}^2) + d(i, b_1) + \Sigma d(e_\imath, b_{\imath+1}) + d(b_K, o)\big).$$

The base case is when there are no segments in $\mathcal{X}^0$ (i.e. $\mathcal{X}^0_{i,o,q} = 0,\ \forall i, o, q$). Here the algorithm just checks that there are no remaining segments in $\mathcal{X}^1$ and $\mathcal{X}^2$. It returns 0 if there are none and ∞ otherwise. This algorithm can be memoized because the number of segments in the first configuration is strictly decreasing.

5.6.2 Correctness

We prove the correctness of this algorithm by induction. The base case is an empty configuration for the parent cluster. The two subconfigurations are therefore compatible with it only if they are also empty. If $\mathcal{X}^0$ is compatible with $\mathcal{X}^1$ and $\mathcal{X}^2$, then the chosen segment of $\mathcal{X}^0$ corresponds by definition to at most Q subsegments in $\mathcal{X}^1$ and $\mathcal{X}^2$. The algorithm enumerates all possible ways to break the segment, so at least one will result in a compatible configuration. Reciprocally, if $\mathcal{X}^0$ is not compatible with $\mathcal{X}^1$ and $\mathcal{X}^2$, the first segment cannot be broken in such a way that results in three compatible configurations, so the algorithm avoids false positives.

5.6.3 Complexity

The complexity of this preprocessing step is

$$O\big(|\{\text{clusters}\}| \cdot |\{\text{configurations}\}|^3 \cdot |\{\text{ways of breaking a segment}\}|\big).$$

We showed in Lemma 13 that there is $|Z|^{O(Q^3\epsilon^{-2}\log^4|Z|)}$ configurations and we showed in the proof of Lemma 14 that there are $O\big((2Q^3\varepsilon^2\log^4|Z|)^Q\big)$ ways of breaking a segment, so the preprocessing can be achieved in $O(|Z| \cdot |Z|^{O\left(Q^3\epsilon^{-2}\log^4|Z|\right)}(2Q^3\varepsilon^{-2}\log^4|Z|)^Q) = O(2^{P(\log|Z|,Q,\varepsilon^{-1})})$ where P is some polynomial.

We can use this preprocessing step to improve the complexity of the main DP. Instead of breaking segments we try all compatible configurations for the children clusters, of which there are at most $O(|\{\text{configuration}\}|^2)$. The complexity of this improved DP is therefore $O(|\{\text{clusters}\}| \times |\{\text{configuration}\}|^3)$ and is dominated by the preprocessing step. Combining this analysis with Theorem 11 gives a QPTAS for planar graphs, proving Theorem 1.

6 Generalizations

6.1 Multiple depots

The techniques presented in this paper can be extended to address the multiple-depot version of Vehicle Routing, assuming a constant number of depots. In this variation, each tour can start and end at different depots. Let R denote the set of depots, and for $v \in Z$ let r_v denote the closest depot to v (note that the tour that visits v does not necessarily visit r_v). The generalization relies on two key observations.

First, the recursive clustering can be slightly modified in the following way. Let a *from-R shortest path* be a from-r shortest path for some $r \in R$.

▶ **Lemma 16.** *Let G be a planar embedded graph with edge costs, and let R and Z be subsets of the vertices. There is a depth-$O(\log|Z|)$ recursive clustering of G with the following properties:*

- *there are no boundary edges,*
- *for each cluster, there are $O(|R|)$ from-R shortest subpaths such that the boundary vertices of the cluster are the vertices that lie on these paths, and*
- *at most three vertices of Z are assigned to each leaf cluster.*

Proof. We sketch the proof. It follows the proof of Lemma 6, which in turn follows that of [8]. Consider the R-rooted shortest-path forest F. Each tree is rooted at some $r \in R$, and consists of those vertices v for which $r_v = r$. Construct a tree T by arbitrarily linking the trees of F, and then use the construction of Lemma 6. This gives a decomposition such that each cluster is bounded by four fundamental cycles of the tree T. Since a fundamental cycle in T consists of at most $2|R|$ from-R shortest paths, this concludes the proof. ◀

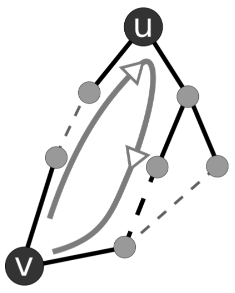


Figure 3 Here, u and v are depots. The forest is in black, the plain lines are the shortest-path trees from u and v and the dashed one is the connecting edge. The dashed, grey lines are edges not in T. The arrow is the boundary of a cluster: there is one fundamental cycle in T, and thus two from-R shortest paths.

Portals in this decomposition are designated the same way as in Section 3. The cost of a detour becomes $\delta(d(v, r_v) + \mathrm{OPT}/Z)$ (using the notation of Section 3), and therefore Lemma 3 has to be adapted in order to obtain an approximate solution.

Each tour P in the optimal solution contains a trip between one depot and the farthest client in P, so the cost of P is at least $\max\{d(c, r_c) : c \text{ is a client of P}\}$, which in turn is at least

$$\frac{1}{Q}\sum\{d(c, r_c) : c \text{ is a client of P}\}$$

by averaging over the at-most Q clients in P.

Using these modified bounds, we can prove a multiple-depot version of the structure theorem, analogous to Theorem 11. Assuming that $|R|$ is constant, we can adapt the dynamic program for this decomposition to get a QPTAS.

6.2 Bounded genus

To extend our algorithm to handle the case when G is embedded on a surface of genus $g > 0$, we adapt a technique previously used by Eistenstat et al. [8]. Let T be any spanning tree of G, in our case the shortest-path tree rooted at the depot. The algorithm selects [9] $2g$ edges not in T such that cutting the surface along the corresponding cycles (each edge forms a cycle with the corresponding simple path in T) yields a surface (with boundary) of genus 0, and a graph embedded on this surface. The vertices of these cycles lie on shortest from-r paths where r is the depot. The algorithm then cuts along these cycles, duplicating the vertices (an edge belonging to such a cycle ends up on one side or the other). The resulting graph is planar. Next the algorithm forms the cluster decomposition for that graph. Finally, the algorithm merges duplicate vertices together. Merging the duplicates can result in those merged vertices being boundary vertices of the clusters, but fortunately all of those merged vertices lie on at most $2g$ from-r shortest paths, so designation of portals can continue as in Section 3.3 and in total the number of portals per cluster will be $O(Qg\epsilon^{-1}\log^2|Z|)$.

6.3 Handling penalties

The dynamic program of Section 5 computes, for each cluster and each admissible configuration of that cluster, the minimum cost partial solution that induces that configuration. To handle penalties, we change the definition of *solution*, of *admissible* and of *cost.* A solution is now allowed to not visit all the clients, an admissible configuration is allowed to not visit all the clients, and the cost includes the penalties of unvisited clients. The base cases change slightly to accommodate these changes, but otherwise the dynamic program is mostly unchanged.

One other change to the algorithm is needed. As described in Section 3.1, the algorithm needs to prune the graph, removing vertices that are too far to be included. To handle penalties, we need a more complicated pruning step. The algorithm computes an upper bound b on the value of the optimum that is at most Q times the value of the optimum, and then prunes away every vertex whose distance from the depot is greater than $b/2$. This ensures that, for any cluster found in the pruned graph, the value $d(r, \hat{v})$ is at most $(Q/2)$OPT, and our analysis can be adapted to show that the number of portals is not too large.

▶ **Lemma 17.** *There exist a polynomial-time algorithm that computes a Q-approximation of the penalty variant of vehicle routing.*

Proof. Consider an instance of the penalty version with capacity Q, and a modified version in which the capacity is 1. Solving the modified instance is easy: for each client, include a depot-to-client round-trip if the cost of this trip is no more than the client's penalty. It remains to show that the optimum value for the original instance is at most Q times the optimum value for the modified instance.

Consider an optimal solution for the original instance. For each tour T in that solution, the cost of T is at least the cost of a round-trip from the depot to the farthest client visited by T. Replace T by a collection of tours, one visiting each of the clients visited by T. Each of these tours is a round trip, and there are Q of them, so their total cost is at most Q times the cost of T. The set of unvisited clients has not changed so the sum of their penalties remains unchanged. Thus the total value of the solution thus obtained is at most Q times the optimum value for the original instance. ◀

References

1 Anna Adamaszek, Artur Czumaj, and Andrzej Lingas. PTAS for k-tour cover problem on the plane for moderately large values of k. *Algorithms and Computation*, pages 994–1003, 2009.

2 Sanjeev Arora, M. Grigni, D. R. Karger, Philip N. Klein, and A. Woloszyn. A polynomial-time approximation scheme for weighted planar graph TSP. In *Proceedings of the 9th Symposium on Discrete Algorithms*, pages 33–41, 1998.

3 Tetsuo Asano, Naoki Katoh, and Kazuhiro Kawashima. A new approximation algorithm for the capacitated vehicle routing problem on a tree. *Journal of Combinatorial Optimization*, 5(2):213–231, 2001.

4 Tetsuo Asano, Naoki Katoh, Hisao Tamaki, and Takeshi Tokuyama. Covering points in the plane by k-tours: a polynomial approximation scheme for fixed k. *IBM Tokyo Research Laboratory Research Report RT0162*, 1996.

5 Tetsuo Asano, Naoki Katoh, Hisao Tamaki, and Takeshi Tokuyama. Covering points in the plane by k-tours: towards a polynomial time approximation scheme for general k. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 275–283. ACM, 1997.

6 Vincent Cohen-Addad, Éric Colin de Verdière, Philip N. Klein, Claire Mathieu, and David Meierfrankenfeld. Approximating connectivity domination in weighted bounded-genus graphs. In *Proceedings of the 48th Annual ACMSymposium on Theory of Computing (STOC*, pages 584–597, 2016. doi:10.1145/2897518.2897635.

7 Aparna Das and Claire Mathieu. A quasipolynomial time approximation scheme for euclidean capacitated vehicle routing. *Algorithmica*, 73(1):115–142, 2015.

8 David Eisenstat, Philip N. Klein, and Claire Mathieu. Approximating k-center in planar graphs. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 617–627. Society for Industrial and Applied Mathematics, 2014.

9 David Eppstein. Dynamic generators of topologically embedded graphs. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 599–608. Society for Industrial and Applied Mathematics, 2003.

10 Bruce L. Golden and Richard T. Wong. Capacitated arc routing problems. *Networks*, 11(3):305–315, 1981.

11 M. Grigni, E. Koutsoupias, and C. Papadimitriou. An approximation scheme for planar graph TSP. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, pages 640–645, 1995.

12 Mark Haimovich and A. H. G. Rinnooy Kan. Bounds and heuristics for capacitated routing problems. *Mathematics of operations Research*, 10(4):527–542, 1985.

13 Shin-ya Hamaguchi and Naoki Katoh. A capacitated vehicle routing problem on a tree. In *International Symposium on Algorithms and Computation*, pages 399–407. Springer, 1998.

14 Stefan Irnich, Paolo Toth, and Daniele Vigo. Chapter 1: The family of vehicle routing problems. In Paolo Toh and Daniele Vigo, editors, *Vehicle Routing: Problems, Methods, and Applications*. SIAM, 2014.

15 Michael Khachay and Roman Dubinin. PTAS for the Euclidean Capacitated Vehicle Routing Problem in R^d. In *Proceedings of the 9th International Conference on Discrete Optimization and Operations Research (DOOR 2016)*, pages 193–205. Springer, 2016.

16 Michael Khachay and Helen Zaytseva. Polynomial time approximation scheme for single-depot euclidean capacitated vehicle routing problem. In *Combinatorial Optimization and Applications*, pages 178–190. Springer, 2015.

17 Mikkel Thorup. Compact oracles for reachability and approximate distances in planar digraphs. *Journal of the ACM*, 51(6):993–1024, 2004.