On matchings, $T$-joins, and arc routing in road networks

Burak Boyacı  |  Thu Huong Dang  |  Adam N. Letchford

1Department of Management Science, Lancaster University, Lancaster, UK
2STOR-i Centre for Doctoral Training, Lancaster University, Lancaster, UK

Correspondence
Thu Huong Dang, STOR-i Centre for Doctoral Training, Lancaster University, Lancaster LA1 4YR, UK.
Email: t.h.dang@lancaster.ac.uk

Funding information
This research was supported by the Routeware, Inc. and EPSRC, Grant/Award Number: EP/L015692/1.

Abstract
Matchings and $T$-joins are fundamental and much-studied concepts in graph theory and combinatorial optimization. One important application of matchings and $T$-joins is in the computation of strong lower bounds for arc routing problems (ARPs). An ARP is a special kind of vehicle routing problem, in which the demands are located along edges or arcs, rather than at nodes. We point out that the literature on applying matchings and $T$-joins to ARPs does not fully exploit the structure of real-life road networks. We propose some ways to exploit this structure. Computational results show significant running time improvements, without deteriorating the quality of the lower bounds.

Keywords
arc routing, combinatorial optimization, matching

1 | INTRODUCTION

Matchings are a fundamental concept in graph theory and combinatorial optimization, with a wide array of applications (see, e.g., [20, 26, 30]). Given an undirected graph $G = (V, E)$, with $|V|$ even, a perfect matching is a set of edges that meets each vertex exactly once. If we are also given a weight $w_e$ for each $e \in E$, the minimum-weight perfect matching problem (WPM for short) calls for a perfect matching of minimum total weight. Edmonds [13] showed that this problem can be solved in polynomial time, and a variety of efficient algorithms are now available (e.g., [12, 19, 26]).

A closely related concept is that of a $T$-join. Given a graph $G$ as before, and a set $T \subseteq V$ with $|T|$ even, a $T$-join is a set of edges that meets each vertex in $T$ an odd number of times, and each other vertex an even number of times. Given edge weights as before, the minimum-weight $T$-join problem (WTJ) calls for a $T$-join of minimum total weight.

Figure 1 illustrates the concept of a $T$-join. On the left is a graph $G$, with nodes in $T$ represented by hollow circles, and weights on the edges. On the right is the same graph, with the MWJ solution indicated with thick lines. (In the online version of this article, the hollow circles and thick lines are in red.)

Edmonds and Johnson [14] showed that any WTJ instance can be transformed into a WPM instance, and thereby solved efficiently. (For details, and alternative algorithms, see Section 2.1.) They also pointed out that $T$-joins can be used to solve a problem that they called the Chinese postman problem (CPP), in honor of Mei-Gu Guan [23].

In the CPP, $G$ represents a road network, and the weight of an edge represents the time taken to traverse the corresponding road. A postman wishes to traverse every road at least once, as quickly as possible, starting and finishing at the same node. Any CPP instance can be reduced to a WTJ instance by setting $T$ to the set of vertices that are incident on an odd number of edges. The edges in the optimal $T$-join then represent roads that need to be traversed twice.

The CPP is an example of an arc routing problem (ARP). An ARP is a special kind of vehicle routing problem, in which the demands are located along edges or arcs, rather than at nodes (e.g., [10]). Whereas the CPP can be solved in polynomial time, most ARPs of interest are $\mathcal{NP}$-hard. Matchings have been used to compute useful lower bounds for such ARPs (e.g., [2, 3, 7, 22, 27, 32-35, 37, 38]).

This is an open access article under the terms of the Creative Commons Attribution License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited.

© 2021 The Authors. Networks published by Wiley Periodicals LLC.
In a recent project with real-life instances, we encountered large-scale ARPs, with over ten thousand edges. For these particular ARPs, matching techniques gave acceptable lower bounds, but used an excessive amount of both time and memory. The purpose of this article is to show that one can dramatically reduce the amount of computational effort needed by matching techniques, by exploiting the structure of real-life road networks. For ease of presentation, we focus on the CPP and another prominent ARP, the so-called *capacitated arc routing problem* or CARP [22].

This article has a simple structure. The key literature is reviewed in Section 2. Some observations concerning road networks are given in Section 3. In Section 4, we show how to solve the CPP more quickly. In Section 5, we show how to compute lower bounds for the CARP more quickly. In Section 6, we make some remarks about possible extensions of our technique to other ARPs.

Throughout this article, all graphs are undirected, simple, and loopless, unless otherwise specified. We let $n$ and $m$ denote $|V|$ and $|E|$, respectively. We also let $\deg(i)$ denote the degree of node $i$ in $G$. We also assume that, in any WTJ instance, all weights are non-negative.

### 2 | LITERATURE REVIEW

We now briefly review the relevant literature. Section 2.1 covers algorithms for WPM and WTJ. Section 2.2 deals with the application of matchings and T-joins to ARPs. Section 2.3 deals with planar graphs.

#### 2.1 | Matchings and T-joins

For surveys of WPM algorithms, see [12, 19, 20, 26, 30]. At present, the fastest strongly polynomial algorithm is that of Gabow [18, 19], which runs in $O(n(m + n \log n))$ time. Among the algorithms that are only weakly polynomial, we mention the one of Duan et al. [12]. It assumes that all weights are integers, and it runs in $O\left(m\sqrt{n} \log(nW)\right)$ time, where $W = \max\{|w_e| : e \in E\}$.

Software for matching has lagged behind the theory to some extent. At present, the most effective available routines for WPM are those of Mehlhorn and Schäfer [31] and Kolmogorov [25]. They perform very well in practice, but run in $O(ncm)$ and $O(n^2m)$ time, respectively.

As for WTJ, Edmonds and Johnson [14] showed that it can be reduced to WPM as follows. First, compute shortest paths between all pairs of nodes in $T$. Then construct a complete undirected “auxiliary” graph, say $G^+$, with $T$ as its node set. Set the weight of each edge in $G^+$ to the length of the corresponding shortest path in $G$. Solve WPM in $G^+$, and then replace each edge in the matching with the corresponding shortest path in $G$. We call this approach “Ed–Jo.” Figure 2 shows how Ed–Jo works for the WTJ instance on the left of Figure 1.
We remark that Dijkstra’s single-source shortest path algorithm can be implemented to run in $O(m + n \log n)$ time [17], and therefore the shortest-path phase in Ed-Jo takes $O(|T| (m + n \log n))$ time. The matching phase takes $O(|T|^3)$ time, since $G^+$ is complete.

Korte and Vygen [26] presented an alternative approach to WTJ, which we will call “Ko-Vy.” Like Ed-Jo, Ko-Vy involves the solution of WPM in an auxiliary graph. However, the auxiliary graph is different. Consider a given node $v \in V$. If $\deg(v)$ is odd and $v \in T$, or $\deg(v)$ is even and $v \notin T$, then $v$ is replaced with a clique on $\deg(v)$ nodes. We call the new nodes “clones”. If $v$ does not satisfy the stated condition, it is replaced by a clique on $\deg(v) + 1$ nodes. We call the additional node a “parity correction” node. The new edges are given a weight of zero.

Figure 3 shows how Ko-Vy works for the WTJ instance in Figure 1. The auxiliary graph is shown on the left, with parity correction nodes represented as green diamonds. An optimal WPM solution is shown on the right.

Ko–Vy is very easy to implement, with no need for shortest-path calculations. On the other hand, the auxiliary graph has $O(m)$ nodes and $O(nm)$ edges. As a result, constructing the auxiliary graph takes $O(nm)$ time and solving the WPM takes $O(n^2 m^2)$ time. More efficient reductions from WTJ to WPM, which use auxiliary graphs with only $O(m)$ nodes and edges, appear in [4, 8]. These reductions enable one to solve WTJ in only $O(m^2 \log n)$ time. They are however trickier to implement.

At the time of writing, the fastest known WTJ algorithm is that of Gabow [19], which runs in $O(|T|(m + n \log n))$ time. It is based on a conversion of WTJ into a capacitated $b$-matching problem. We omit details for brevity.

2.2 Applications to arc routing

We mentioned above that the CPP can be converted to WPM or WTJ. The authors have also used WPM and/or WTJ to compute lower bounds for $\mathcal{NP}$-hard ARPs (e.g., [2, 3, 27, 33, 34]). For brevity, we focus here on the CARP [22], which has received the most attention.
In the CARP, we are given a graph \( G = (V, E) \) and a set \( E_R \subseteq E \) of required edges. Node 0 represents the depot. For each \( e \in E \), we are given a positive cost \( c_e \). For each \( e \in E_R \), we are given a positive demand \( q_e \). An unlimited fleet of identical vehicles, each of capacity \( Q \), is located at the depot. The task is to find a minimum-cost set of routes, each starting and ending at the depot, such that each required edge is serviced on exactly one route, and the total demand on each route does not exceed \( Q \).

Many lower bounds for the CARP have been proposed which use WPM as a sub-routine [3, 7, 22, 32, 35, 37, 38]. For brevity, we review only one in detail: the bound called “LB1” in [7]. To describe it, we need some more notation. We let \( AHR \) [1] showed that NDLB is slightly stronger than LB1, though this comes at the cost of a significantly increased running time.

Among the other matching-based bounds for the CARP, we mention only the node duplication lower bound (NDLB) from [35]. The procedure is very similar to the one for LB1, but the auxiliary graph is much larger than \( G \), with up to \( 2(|E_R|+K) \) nodes. Ahr [1] showed that NDLB is slightly stronger than LB1, though this comes at the cost of a significantly increased running time.

### 2.3 Planar graphs

Now we recall some facts concerned with planar graphs. The first is Euler’s theorem, which states that, in a planar graph, \( m \leq 3n - 6 \). This implies that, when \( G \) is planar, one can solve WPM in \( O(n^2 \log n) \) time.

Lipton and Tarjan [29] showed that, in fact, it is possible to solve planar WPM in only \( O(n^{3/2} \log n) \) time. Their algorithm exploits a famous result in their earlier paper [28], which states that, in any planar graph, there exists a set \( S \subseteq V \) such that (a) \( |S| = O(\sqrt{n}) \) and (b) if \( S \) is removed from \( G \), each connected component in the resulting graph contains no more than \( 2n/3 \) nodes. A suitable \( S \), called a separator, can be found in linear time [28].
Henzinger et al. [24] gave an algorithm for single-source shortest-paths in planar graphs, which runs in only $O(n)$ time. This implies that the shortest-path phase of Ed-Jo algorithm can be conducted in only $O(n|T|)$ time in the planar case. The matching phase, however, still takes $O(|T|^3)$ time.

It is also worth noting that the reduction from WTJ to WPM in Barahona [4] preserves planarity. Together with the Lipton-Tarjan result, this implies that planar WTJ can be solved in $O(n^{3/2} \log n)$ time. A more direct algorithm, with the same running time, is given in Barahona [5].

To close this section, we mention that road networks are often planar and, even if not, they invariably contain small separators. Fast algorithms for finding small separators in road networks, together with encouraging computational results, are given in [11] and [36].

### 3 | ON ROAD NETWORKS

In this section, we make some observations regarding real-life road networks. Our first observation is that road networks are not just sparse; they also have bounded degree. Although this is obvious intuitively, we decided to compute the degree distribution for a sample of 12 cities from across the world, using the Python package OSMnx [9]. For each city, we considered the closest 1000 nodes to a central landmark, and computed their degrees. The results are shown in Table 1.

As one might expect, the vast majority of nodes have degree 3 or 4. We expected the maximum degree to be 5, but we see instead that it is 7. On the other hand, only 3 nodes (out of 12 000) have degree 7. We remark that the average degree ranged from 2.71 (Hanoi) to 3.71 (New York).

The reason that this is relevant is that, in Ko-Vy, a node of degree $d$ in $G$ becomes a clique of size $d$ in the auxiliary graph (or $d + 1$ if a parity correction node is needed). If $G$ has bounded degree, the size of each clique in the auxiliary graph will be $O(1)$, and the auxiliary graph will have only $O(n)$ nodes and edges. Not only that, but Ko-Vy is much easier to implement than either Ed-Jo or the methods in [4] and [8]. Thus, for road networks, Ko-Vy may be an attractive alternative to those methods. We will see in the next section that this is indeed the case.

Next, we considered the issue of planarity. For each city, we computed the graph induced by the given 1000 nodes. We then tested planarity using a package called Python Planarity\(^1\). We were surprised to find that none of the graphs were planar. Closer inspection revealed that all of them could be made planar by deleting a very small number of edges (corresponding to over- or under-passes).

Finally, we decided to test the claim made in [11] and [36] that road networks tend to have small separators. We found that, indeed, all 12 graphs contained separators of size $O(\sqrt{n})$. Moreover, such separators can be found easily. One way is as follows. Let $G = (V, E)$ be the graph in question, and let $G^-$ be a planar subgraph obtained by deleting some edges corresponding to over- or under-passes. Compute a separator $S \subseteq V$ in $G^-$ using the linear-time algorithm in [28]. By definition, removing $S$ from $V$ causes $G^-$ to become disconnected. Let $F$ be the set of edges in $E$ that have end-nodes in different connected components. If $F = \emptyset$, we are done. Otherwise, use a greedy algorithm to construct a minimal set of nodes $T \subseteq V \setminus S$ that covers the edges in $F$. By definition, $S \cup T$ is a separator in $G$.

\(^1\)https://anaconda.org/conda-forge/python-planarity
We conjecture that the planar WTJ algorithm of Barahona [5] could be modified, using the small separators mentioned, to solve the CPP in road networks in $O(n^{3/2} \log n)$ time.

### 4 The Chinese Postman Problem in Road Networks

Armed with the facts mentioned in the previous section, we can now analyze the theoretical running times of various approaches to the CPP in road networks:

1. **Ed-Jo**: The number of shortest-path calls is $|V_0|$, and each call takes $O(n \log n)$ time. So the shortest-path phase takes $O(|V_0| n \log n)$ time. The auxiliary graph is complete and has $|V_0|$ nodes, so the matching phase takes $O(|V_0|^3)$ time.
2. **Ko-Vy**: Given that road networks have bounded degree, Korte and Vygen’s auxiliary graph takes only $O(n)$ time to construct. Moreover, the auxiliary graph contains only $O(n)$ nodes and edges. As a result, the matching phase takes $O(n^2 \log n)$ time.
3. The approaches in [4] and [8]. These also yield auxiliary graphs with $O(n)$ nodes and edges. Thus, they take the same time (asymptotically) as Ko-Vy.

The above analysis suggests that Ko-Vy should be faster than Ed-Jo when applied to large-scale CPP instances on road networks. To test this, we created 18 CPP instances, with cities selected from Paris, London and Moscow, and with $n$ selected from 1000, 2000, 5000, 10 000, 20 000, and 50 000. The nodes were selected as in the previous subsection, and all edges having both end-nodes in the given set of nodes were put into the graph. Note that none of the resulting 18 graphs were planar. The cost of each edge was set to the length of the corresponding road, rounded to the nearest meter. The data for each instance, along with the optimal solution values, is made available at the Lancaster University Data Repository².

We remark that some of the graphs contained loops (edges that connect a node with itself). Most of these loops represented small areas, near entrances of hotels, where taxis can change direction. We removed them, for simplicity of implementation.

For Ed-Jo, we implemented our own version of Dijkstra’s single-source shortest path algorithm. We used a binary heap, since it is much easier to code than a Fibonacci heap, yet its running time is $O(n \log n)$ for sparse graphs. To solve the WPM instances, we used the open-source software package Blossom V [25]. Although it runs in $O(n^2 m)$ time in theory, it performs extremely well in practice, as we will see below. For all experiments, we used a Lenovo ThinkPad laptop with an i5-8250U processor, running under Windows 10 at 1.6 GHz with 16 GB of RAM.

Table 2 shows the results obtained when transforming each of the 18 CPP instances into WPM instances. For each instance, we show the name of the city, the number of nodes ($n$), and the number of edges ($m$). Then, for each approach, we show the number of nodes ($\bar{n}$) and edges ($\bar{m}$) in the resulting auxiliary graph. It is clear that Ed-Jo leads to auxiliary graphs with far fewer nodes than Ko-Vy. On the other hand, the number of edges is drastically bigger.

Table 3 shows, for each instance and each approach, the time taken to construct the auxiliary graph (T1) and the time taken to solve the WPM instance (T2). All times are reported in seconds. A dash indicates that the WPM phase had to be aborted due to memory limitations.

It is clear that the traditional approach, Ed-Jo, is too slow to be used for large-scale instances. In particular, the shortest-path phase is time-consuming. In principle, one could obtain a speed-up by computing the shortest paths in parallel. Nevertheless, the matching phase of Ed-Jo is itself rather slow. Moreover, we found that the matching phase consumed a great deal of memory for these instances.

We were surprised to find that the other approach, Ko-Vy, is orders of magnitude faster. Indeed, it takes less than one second for every instance. This is no doubt due to the fact that road networks have bounded degree.

For interest, we also implemented a third reduction from WTJ to WPM, due to Barahona [4]. The resulting auxiliary graphs were a bit larger than the ones that we obtained with Ko-Vy. As a result, the WPM phase took about twice as long to solve. We omit details for brevity. The main conclusion from this section is that, for large-scale CPP instances on road networks, Ko-Vy is preferable to Ed-Jo.

### 5 The Capacitated Arc Routing Problem in Road Networks

We now move on to consider the CARP. We start by analyzing the time taken to compute LB1 in the case of road networks. The procedure begins by solving $|V_0|$ shortest-path problems in $G$. Since $G$ is sparse, this takes $O(|V_0| n \log n)$ time. The next step

²http://www.research.lancs.ac.uk/portal/en/datasets/search.html
is to construct the auxiliary graph, which we will call \( \tilde{G} \). Note that \( \tilde{G} \) has \( O(|V_O| + K) \) nodes, and therefore solving the WPM takes \( O(|V_O|^3 + K^3) \) time. This is \( O(n^3) \) in the worst case.

We now present a procedure that yields lower bounds of comparable quality to those of LB1, but which exploits the properties of road networks. The procedure incorporates concepts from both LB1 and Ko-Vy.

The first steps in our algorithm are identical to that of LB1. That is, we compute \( K \) (the lower bound on the number of vehicles), the shortest paths in \( G \) from the depot to each node in \( V_R \), the values \( r_1, \ldots, r_s \), and \( s \) (the smallest integer such that \( r_1 + \cdots + r_s \geq 2K \)). We then let \( V_o \) denote \( \{v_1, \ldots, v_s\} \).

The next step is different. We take \( G \) and apply the Korte-Vygen procedure with \( T \) set to \( V_O \), the set of nodes that are incident on an odd number of required edges. The resulting graph will be called \( G^\ast \). Figure 5 shows the graph \( G^\ast \) for the CARP instance on the left of Figure 4. We remind the reader that \( K = 2 \) and \( s = 3 \) for this example.
As before, we call the copies of the original nodes “clones,” and any additional nodes “parity correction” nodes. If a clone is incident on a required edge, it will be called an “R-clone.” Note that, if a node in \( G \) is incident on \( t \) required edges, then \( t \) of its clones will be R-clones. In particular, there are \( r_i \) R-clones of node \( v_i \). Note also that, since \( G \) has bounded degree, \( G^+ \) has only \( O(n) \) nodes and edges.

For a given node \( v \in V \), we will call the corresponding set of nodes in \( G^+ \) a “gadget,” and denote it by \( g(v) \). In Figure 5, each gadget is enclosed in a circle. We also let \( g'(v) \) denote the corresponding set of nodes in \( \tilde{G} \), the auxiliary graph that is used for LB1 (see Section 2.2). Note that \(|g(v)| \geq \deg(v) \geq \deg_g(v) \geq |g'(v)|\) for every \( v \in V \). Moreover, \(|g(v)| - |g'(v)|\) is always even. (Indeed, \( \deg_g(v) \) and \( \deg(v) \) have the same parity if and only if there is no parity correction node in \( g(v) \).)

We now create an even bigger graph, called \( G^{++} \). We begin by taking \( G^+ \) and adding \( 2K \) extra “dummy” nodes, representing copies of the depot. We will call the dummy nodes \( D_1, D_2, \ldots, D_{2K} \). We then add several sets of edges:

- We connect \( D_1 \) to the first R-clone of \( v_1 \), \( D_2 \) to the second R-clone of \( v_1 \), and so on. In this way, we connect each dummy node to its own R-clone. The weight of each additional edge is the length of the shortest path from the depot to the corresponding member of \( V_c \).
- For each \( i \in V_O \setminus V_c \), we pick one R-clone as a “representative” of \( i \). We then add an edge between each dummy node and each representative. The weight of each edge is the length of the shortest path from the depot to the given node \( i \).
- Now let \( p = \sum_{i=1}^{s-1} r_i - 2K \). If \( p > 0 \), we have \( p \) R-clones of \( v_i \) that are not connected to any dummy nodes. We pick one of those as a “representative,” and connect each dummy node to it. The weight of each additional edge is the length of the shortest path from the depot to \( v_i \).

Figure 6 shows the graph \( G^{++} \) for the same CARP instance as before. Since \( K = 2 \), there are four dummy nodes. We have \( s = 3 \) and \( V_c = \{v_1, v_2, v_3\} \). Thus, two dummy nodes are connected to R-clones of \( v_1 \), one is connected to the R-clone of \( v_2 \), and the other is connected to an R-clone of \( v_3 \). We also have \( V_O \setminus V_c = \{v_4\} \). Thus, all dummy nodes are connected to a representative of \( v_4 \). Finally, we have \( p = 5 - 4 = 1 \). Since \( p \) is positive and \( s = 3 \), all dummy nodes are connected to a representative of \( v_3 \) (namely, the unique R-clone of \( v_3 \) that was not already connected to a dummy node).

Note that, when \( G \) is sparse and has bounded degree, \( G^{++} \) has only \( O(n) \) nodes and \( O(n + K |V_O|) \) edges. We will show that the solution to the WPM in \( G^{++} \) yields a valid lower bound for the CARP. We will need the following lemma.

**Lemma 1.** Let \( p' = 2K - \sum_{i=1}^{s-1} r_i \) and note that \( 1 \leq p' \leq r_s \). In \( \tilde{G} \), there exists a minimum-weight perfect matching in which no more than \( p' \) dummy nodes are matched with R-clones of \( v_i \).

**Proof.** Let \( M \) be a perfect matching in \( \tilde{G} \) that matches \( k \) dummy nodes with R-clones of \( v_i \), where \( k > p' \). We will show that we can construct a perfect matching \( M' \), of no larger cost, such that \( k - 2 \) dummy nodes are matched with R-clones of \( v_i \). Let \( j_1 \) and \( j_2 \) be two R-clones of \( v_i \) that are currently matched with dummy nodes. We assume w.l.o.g. that those dummy nodes are \( D_1 \) and \( D_2 \). Also let \( I \) denote the set of R-clones of nodes in \( V_c \setminus \{v_i\} \) that are currently not matched with dummy nodes, and note that \(|I| \geq k \geq 2 \). We consider two cases:

Case 1: there exist two R-clones in \( I \), say \( i_1, i_2 \), that were matched in \( M \). We obtain a matching of no larger cost by deleting the edges \( \{D_1, j_1\}, \{D_2, j_2\} \) and \( \{i_1, i_2\} \), and adding the edges \( \{j_1, i_2\}, \{D_1, i_1\} \) and \( \{D_2, i_2\} \) (see Figure 7). To see why, note that the nodes in \( V_c \) represented by \( i_1 \) and \( i_2 \) are no further from the depot than \( v_i \).
Case 2: no pair of R-clones in \( I \) were matched in \( M \). In this case, the R-clones in \( I \) must be matched with R-clones of nodes in \( V_O \setminus V_c \). Let \( i_1 \) be an R-clone in \( I \), and \( i_2 \) be an R-clone of a node in \( V_O \setminus V_c \), such that \( i_1 \) and \( i_2 \) are matched in \( M \). As before, we obtain a matching of no larger cost by deleting the edges \( \{D_1, j_1\}, \{D_2, j_2\} \) and \( \{i_1, i_2\} \) and adding the edges \( \{j_1, j_2\}, \{D_1, i_1\}, \) and \( \{D_2, i_2\} \). To see why, note that the cost of the shortest path from the depot to the node represented by \( i_2 \) is no larger than the cost of any path that passes through the node represented by \( i_1 \).

\[ \square \]

**Theorem 1.** A valid lower bound for the CARP is obtained by solving WPM in \( G^{++} \), and then adding the cost of the required edges.

**Proof.** Recall that \(|g(v)| - |g'(v)| \geq 0\) and even for every \( v \in V \). Let \( M \) be a perfect matching in \( G \). We construct a perfect matching \( M' \) in \( G^{++} \), as follows:

- We first match the dummy nodes. If \( k \) R-clones of a given node \( v \) are matched with dummy nodes in \( \tilde{G} \), we match \( k \) corresponding R-clones of \( v \) with dummy nodes in \( G^{++} \). We can apply this procedure until all dummy nodes are matched since, by Lemma 1, there are no more than \( 2K - \sum_{i=1}^{r_v} \) R-clones of \( v_i \) matched with dummy nodes in \( \tilde{G} \).
- We then take the other edges in \( M \), if any, and construct the corresponding edges in \( M' \). For a matched edge \( \{i, j\} \) in \( \tilde{G} \), we match in \( G^{++} \) pairs of clones of the nodes in the shortest path between the corresponding nodes in \( G \). We can repeat this until all matched edges in \( \tilde{G} \) are considered since, for any node \( v \in V \), the gadget \( g(v) \) contains at least \( \deg(v) \) clones.
- Finally, we check to see if there are any unmatched nodes in \( G^{++} \). Consider a node \( v \in V \). By construction, the number of unmatched copies of \( v \) in \( G^{++} \), if any, must be even. Thus, if such copies of \( v \) exist, we can match pairs of them using edges in \( G^{++} \) of zero cost.

By construction, \( M \) and \( M' \) have the same cost. \[ \square \]

Figure 8 illustrates Theorem 1 for our CARP example. The graph \( G^{++} \) is shown once more, but with an optimal WPM solution indicated by thick red lines. The weight of the matching is 37, so the resulting lower bound is \( 37 + 18 = 55 \). We remark that LB1 has the same value for this example.

To test the new bounding procedure, we converted each of our 18 CPP instances into CARP instances, as follows. The depot is located near the center. Edges incident on the depot were made non-required. Each other edge was made required with probability \( 1/2 \). The demand of each required edge was set to a random integer between 1 and 10. To make the fleet size
As before, we used Blossom V to solve the matching problems. For the largest instances, with 50 000 nodes, our PC ran into memory difficulties. Accordingly, we used a workstation with Intel Xeon E5-2640 v3 processor at 2.6 GHz and 32 GB of RAM for those instances.

Table 4 gives information concerning the size of the auxiliary graphs. It is clear that our transformation leads to auxiliary graphs with more nodes, but the number of edges is dramatically smaller.

Table 5 shows, for each instance and each approach, the time taken to construct the auxiliary graph (T1), the time taken to solve the WPM instance (T2), and the resulting lower bound. One can see that, for these instances, our procedure is much faster than the one in [7]. Moreover, our lower bound matches LB1 in 16 out of 18 cases, and is only slightly weaker in the remaining two cases.

For interest, we also implemented NDLB from [35]. The resulting auxiliary graphs were much larger than the ones from either LB1 or our procedure, and they took much longer to compute. The lower bounds were slightly better than LB1, by around 5% on average, but the WPM phase took around ten times longer than it did for LB1. We also experienced even more problems with memory. We omit details for brevity.
We close this section with a remark. In our approach, we use $2K$ dummy nodes. It would be much more efficient to use just one dummy node, and require it to have degree $2K$. One would then have to solve a minimum-weight $f$-factor problem in $G^{++}$, instead of a WPM. Unfortunately, at the time of writing, no efficient open-source $f$-factor code was available.

## 6 CONCLUDING REMARKS

We have shown that, by exploiting the special structure of road networks, we can solve large-scale CPP instances in less than a second, and compute strong lower bounds for large-scale CARP instances within a few seconds. Our procedures are orders of magnitude faster than previous approaches in the literature. Moreover, any future improvements in matching software will make our procedures even better.

We believe that our approach could be fairly easily adapted to other ARPs, such as ARPs with multiple depots [15], time deadlines, [16, 27] and intermediate facilities [21], or ARPs on mixed graphs [6].

## ACKNOWLEDGEMENTS

The second author gratefully acknowledges financial support from Routeware, Inc., and from the EPSRC through the STOR-i Centre for Doctoral Training under grant EP/L015692/1.

## ORCID

Burak Boyacı  https://orcid.org/0000-0001-8392-3078

Thu Huong Dang  https://orcid.org/0000-0003-2184-191X

Adam N. Letchford  https://orcid.org/0000-0002-3191-5006

## REFERENCES

[1] D. Ahr. Contributions to multiple postmen problems, Ph.D. thesis, Institut für Mathematik, Heidelberg University, 2004.
[2] D. Ahr and G. Reinelt, *New heuristics and lower bounds for the min-max k-Chinese postman problem*, in *Proceedings of ESA ’02*, R. Möhring and R. Raman, Eds., Springer, Heidelberg, Germany, 2002, 64–74.
[3] D. Ahr and G. Reinelt, *The capacitated arc routing problem: Combinatorial lower bounds*, in *Arc routing: Problems, methods, and applications*, Á. Corberán and G. Laporte, Eds., SIAM, Philadelphia, PA, 2014, 159–181.
[4] F. Barahona, *On the computational complexity of Ising spin glass models*, J. Phys. A 15 (1982), 3241–3253.
B. Raghavachari and J. Veerasamy, A. Schild and C. Sommer, Y. Saruwatari, R. Hirabayashi, and N. Nishida, Contributions to routing problems, Ph.D. thesis, Institut für Mathematik, Augsburg University, 1987.

B. Raghavachari and J. Veerasamy, K. Mehlhorn and G. Schäfer, R. J. Lipton and R. E. Tarjan, L. Lovász and M. D. Plummer, R. J. Lipton and R. E. Tarjan, L. Y. O. Li, Vehicle routeing for winter gritting, Ph.D. thesis, Department of Management Science, Lancaster University, 1992.

B. Korte and J. Vygen, V. Kolmogorov, M. R. Henzinger, P. Klein, S. Rao, and S. Subramanian, M.-G. Guan, B. L. Golden and R. T. Wong, A. H. M. Gerards, G. Ghiani, G. Improta, and G. Laporte, Eds., SIAM, Philadelphia, PA, 1999, 734–741.

R. Duan, S. Pettie, and H.-H. Su, Scaling algorithms for weighted matching in general graphs, ACM Trans. Algorithms 14 (2018), 1–35.

J. Edmonds, Matching, Euler tours and the Chinese postman, Math. Program. 5 (1973), 88–124.

R. W. Eglese and L. Y. Li, A tabu search based heuristic for arc routing with a capacity constraint and time deadline, in Meta-heuristics: Theory and applications, I. H. Osman and J. P. Kelly, Eds., Kluwer, Boston, MA, 1996, 633–649.

M. L. Fredman and R. E. Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms, J. ACM 34 (1987), 596–615.

H. N. Gabow, Data structures for weighted matching and extensions to b-matching and f-factors, ACM Trans. Algorithms 14 (2018), 1–80.

H. N. Gabow, Data structures for weighted matching and nearest common ancestors with linking, in Proceedings of SODA ‘90, D. Johnson, Ed., SIAM, Philadelphia, PA, 1990, 434–443.

H. N. Gabow, Data structures for weighted matching and near common ancestors with linking, in Proceedings of SODA ’90, D. Johnson, Ed., SIAM, Philadelphia, PA, 1990, 434–443.

Ghiani, G. Improta, and G. Laporte, The capacitated arc routing problem with intermediate facilities, Networks 37 (2001), 134–143.

B. L. Golden and R. T. Wong, Capacitated arc routing problems, Networks 11 (1981), 305–315.

M.-G. Guan, Graph partitioning using odd or even points, Chinese Math. 1 (1962), 273–277.

M. R. Henzinger, P. Klein, S. Rao, and S. Subramanian, Faster shortest-path algorithms for planar graphs, J. Comput. Syst. Sci. 55 (1997), 3–23.

V. Kolmogorov, Blossom V: A new implementation of a minimum cost perfect matching algorithm, Math. Program. Comput. 1 (2009), 43–67.

B. Korte and J. Vygen, Combinatorial optimization: Theory and algorithms, 6th ed., Springer, Heidelberg, Germany, 2018.

L. Y. O. Li, Vehicle routing for winter gritting, Ph.D. thesis, Department of Management Science, Lancaster University, 1992.

R. J. Lipton and R. E. Tarjan, A separator theorem for planar graphs, SIAM J. Appl. Math. 36 (1979), 177–189.

R. J. Lipton and R. E. Tarjan, Applications of a planar separator theorem, SIAM J. Comput. 9 (1980), 615–627.

L. Lovász and M. D. Plummer, Matching theory, AMS, Providence, RI, 2009.

K. Mehlhorn and G. Schäfer, Implementation of $O(nm \log n)$ weighted matchings in general graphs: The power of data structures, J. Exp. Algorithmics 7 (2002), 4.

W. L. Pearn, New lower bounds for the capacitated arc routing problem, Networks 18 (1988), 181–191.

B. Raghavachari and J. Veerasamy, A 3/2-approximation algorithm for the mixed postman problem, SIAM J. Discret. Math. 12 (1999), 425–433.

B. Raghavachari and J. Veerasamy, Approximation algorithms for the asymmetric postman problem, in Proceedings of SODA ’99, R. E. Tarjan and T. Warnow, Eds., SIAM, Philadelphia, PA, 1999, 734–741.

Y. Saruwatari, R. Hirabayashi, and N. Nishida, Node duplication lower bounds for the capacitated arc routing problem, J. Oper. Res. Soc. Jpn. 35 (1992), 119–133.

A. Schlich and C. Sommer, On balanced separators in road networks, in Proceedings of SEA ’15, E. Bampis, Ed., Springer, Heidelberg, Germany, 2015, 286–297.

Z. Win, Contributions to routing problems, Ph.D. thesis, Institut für Mathematik, Augsburg University, 1987.

S. Wohlk, New lower bound for the capacitated arc routing problem, Comput. Oper. Res. 33 (2006), 3458–3472.

How to cite this article: Boyacı B, Dang TH, Letchford AN. On matchings, T-joins, and arc routing in road networks. Networks. 2022;79:20–31. https://doi.org/10.1002/net.22033