On Matchings, T-Joins, and Arc Routing in Road Networks

Burak Boyacı[*] Thu Huong Dang[†] Adam N. Letchford[*]

To appear in *Networks*

Abstract

Matchings and *T-joins* are fundamental and much-studied concepts in graph theory and combinatorial optimisation. 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: combinatorial optimization; arc routing; matching

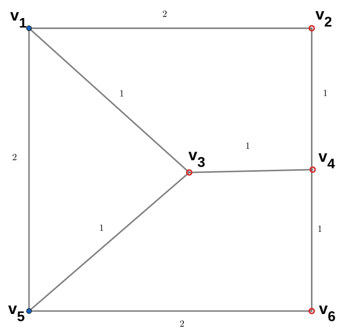
1 Introduction

Matchings are a fundamental concept in graph theory and combinatorial optimisation, 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]).

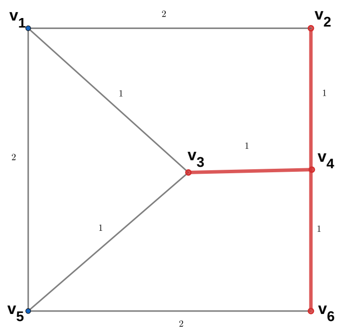
A closely related concept is that of a *T-join.* Given a graph G as before, and a set $T \subseteq V$ with $|T|$ even, a T-join is a set of edges that meets each vertex in T an odd number of times, and each other vertex an even number of times. Given edge weights as before, the *minimum-weight T-join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

[*]Department of Management Science, Lancaster University, Lancaster LA1 4YX, UK. E-mail: {B.Boyaci,A.N.Letchford}@lancaster.ac.uk

[†]STOR-i Centre for Doctoral Training, Lancaster University, Lancaster LA1 4YR, UK. E-mail: T.H.Dang@lancaster.ac.uk



(a) Graph G with nodes in T indicated



(b) Graph with minimum weight T-join

Figure 1: An example of the original graph

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

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 Subsection 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 honour 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, 33, 34, 35, 37, 38]).

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 paper 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].

The paper 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 the paper, 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. Subsection 2.1 covers algorithms for WPM and WTJ. Subsection 2.2 deals with the application of matchings and T-joins to ARPs. Subsection 2.3 deals with planar graphs.

2.1 Matchings and T-joins

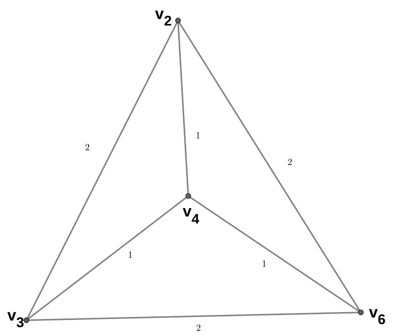
For surveys of WPM algorithms, see [12, 19, 20, 26, 30]. At present, the fastest strongly polynomial algorithm is that of Gabow [18, 19], which runs in $O(n(m + n \log n))$ time. Among the algorithms that are only weakly polynomial, we mention the one of Duan *et al.* [12]. It assumes that all weights are integers, and it runs in $O(m\sqrt{n} \log(nW))$ 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 $O(nm \log n)$ and $O(n^2m)$ time, respectively.

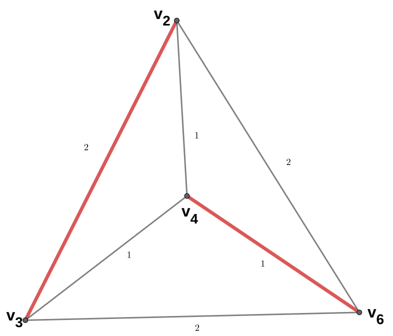
As for WTJ, Edmonds and Johnson [14] showed that it can be reduced to WPM as follows. First, compute shortest paths between all pairs of nodes in T. Then construct a complete undirected "auxiliary" graph, say G^+, with T as its node set. Set the weight of each edge in G^+ to the length of the corresponding 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 & 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

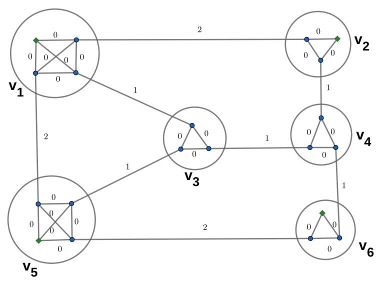


(a) Auxiliary graph G^+

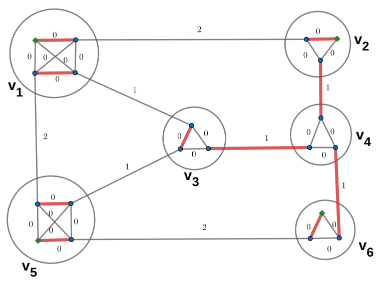


(b) An optimal WPM solution

Figure 2: Ed-Jo transformation



(a) Auxiliary graph



(b) WPM solution

Figure 3: Ko-Vy transformation

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 $O(n^2m^2)$ 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 more tricky 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

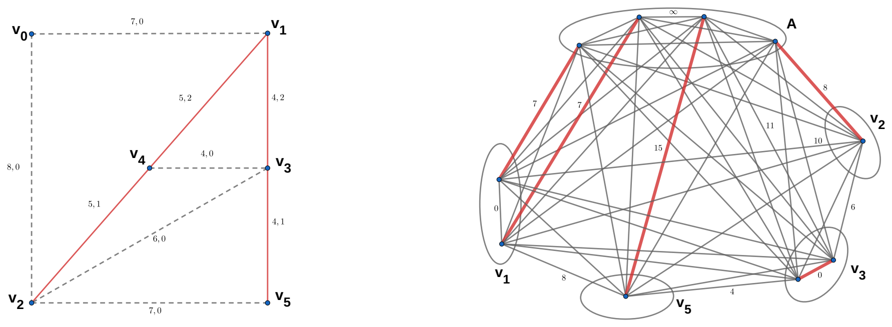
We mentioned above that the CPP can be converted to WPM or WTJ. Authors have also used WPM and/or WTJ to compute lower bounds for $\mathcal{NP}$-hard ARPs (e.g., [2, 3, 27, 33, 34]). For brevity, we focus here on the CARP [22], which has received the most attention.

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 V_R and V_O denote the set of nodes in G that are incident on at least one required edge, and on an odd number of required edges, respectively. We also let K denote $\lceil \sum_{e \in E_R} q_e / Q \rceil$, which is a lower bound on the number of routes. Finally, for notational simplicity, we assume that no required edges are incident on the depot. (If this is not the case, then we can make it so by adding a dummy node to G, along with a dummy edge of zero cost.)

LB1 works as follows. First, compute shortest paths from the depot to each node in V_R. Let v_1 be the closest node to the depot, v_2 be the second closest, and so on. Let r_i be the number of required edges incident on node v_i. Let s be the smallest integer such that $r_1 + \cdots + r_s \geq 2K$, and let V_c denote $\{v_1, \ldots, v_s\}$. (The subscript c is to remind us that the nodes in V_c are "close" to the depot). Let A consist of $2K$ "dummy" nodes, representing copies of the depot. Let B contain r_i "clones" of vertex v_i, for $i = 2, \cdots, s$. Let S contain all nodes in $V_O \setminus V_c$, except the depot. Construct a complete graph, say $\bar{G}$, with node set $A \cup B \cup S$. The weight of each edge in $\bar{G}$ is set to the cost of the shortest path between the corresponding nodes in G, with one exception: edges between dummy nodes are given infinite weight. Finally, solve the WPM in $\bar{G}$. Then LB1 is the weight of the WPM solution plus the cost of the required edges.

The above-mentioned procedure is illustrated in Figure 4. On the left, we see a graph G, with required and non-required edges represented by red lines and grey dashed lines, respectively. For each edge $e \in E$, the cost c_e



(a) Graph **(b) WPM solution in auxiliary graph**

Figure 4: LB1 approach

and demand q_e are indicated. We suppose that $Q = 4$ and that the depot is node v_0. Note that $K = 2$ and $s = 3$ for this instance. On the right, we show the auxiliary graph $\bar{G}$, with an optimal WPM solution indicated by thick red lines. The weight of the matching is 37, which yields a lower bound of $37 + 18 = 55$.

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 $\bar{G}$, 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 $S \subset V$ 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, 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 twelve 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.

City	1	2	3	4	5	6	7
Barcelona	38	5	527	418	12	0	0
Hanoi	208	15	635	140	2	0	0
Istanbul	34	29	700	235	2	0	0
Johannesburg	51	14	542	386	5	2	0
Karachi	33	4	733	226	4	0	0
London	111	34	649	202	4	0	0
Madrid	68	12	682	234	4	0	0
Mexico City	68	18	598	312	4	0	0
Moscow	51	51	734	183	8	1	1
New York	10	6	261	709	12	2	0
Paris	50	21	646	261	29	2	1
Seoul	86	6	786	116	5	0	1

Table 1: Number of nodes having a given degree for twelve cities.

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, 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, 36] that road networks tend to have small separators. We found that, indeed, all twelve 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 $S \subset V$ 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 $T \subset V \setminus S$ that covers the edges in F. By definition, $S \cup T$ is a separator in G.

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 analyse the theoretical running times of various approaches to the CPP in road networks:

1. `Ed-Jo`. The number of shortest-path calls is $|V_O|$, and each call takes $O(n \log n)$ time. So the shortest-path phase takes $O(|V_O|\, n \log n)$ time. The auxiliary graph is complete and has $|V_O|$ nodes, so the matching phase takes $O(|V_O|^3)$ 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, 8]. These also yield auxiliary graphs with $O(n)$ nodes and edges. Thus, they take the same time (asymptotically) as `Ko-Vy`.

[1]`https://anaconda.org/conda-forge/python-planarity`

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, 10000, 20000 and 50000. 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 Repository[2].

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 16GB 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 ($\tilde{n}$) and edges ($\tilde{m}$) 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 `Blossom V` had to be aborted due to memory limitations.

It is clear that the traditional approach, `Ed-Jo`, is to 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.

[2]`http://www.research.lancs.ac.uk/portal/en/datasets/search.html`

	Original Graph		Ed-Jo		Ko-Vy	
City	n	m	$\tilde{n}$	$\tilde{m}$	$\tilde{n}$	$\tilde{m}$
London	1000	2016	432	93096	4032	9294
	2000	4209	812	329266	8418	20052
	5000	11197	1658	1373653	22394	56941
	10000	22833	2712	3676116	45666	118800
	20000	46948	4214	8876791	93896	251351
	50000	118345	7584	28754736	236690	642368
Moscow	1000	1872	564	158766	3744	7746
	2000	3848	1084	586986	7696	16334
	5000	10136	2606	3394315	20272	45925
	10000	20773	4948	12238878	41546	96736
	20000	42410	8900	39600550	84820	203976
	50000	115464	12180	74170110	230928	616878
Paris	1000	1992	576	165600	3984	8878
	2000	3951	1140	649230	7902	17412
	5000	9780	2878	4140003	19560	42291
	10000	19707	5638	15890703	39414	85957
	20000	40499	10096	50959560	80998	182386
	50000	105791	21352	227943276	211582	503268

Table 2: Effect of transformations on graph size.

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 analysing the time taken to compute LB1 in the case of road networks. The procedure begins by solving $|V_O|$ shortest-path problems in G. Since G is sparse, this takes $O(|V_O|\, n \log n)$ time. The next step 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.

		Ed-Jo		Ko-Vy	
City	n	T1	T2	T1	T2
London	1000	0.266	0.040	0.005	0.007
	2000	1.237	0.267	0.004	0.010
	5000	7.217	1.645	0.018	0.026
	10000	24.316	10.162	0.065	0.056
	20000	89.500	13.877	0.178	0.101
	50000	466.152	55.507	0.456	0.216
Moscow	1000	0.337	0.137	0.007	0.004
	2000	1.652	0.564	0.005	0.015
	5000	11.325	3.886	0.023	0.023
	10000	48.196	16.587	0.030	0.056
	20000	210.821	68.541	0.237	0.106
	50000	788.479	142.496	0.463	0.197
Paris	1000	0.348	0.180	0.005	0.005
	2000	2.061	0.712	0.004	0.013
	5000	11.895	7.463	0.009	0.038
	10000	54.263	38.424	0.098	0.105
	20000	252.912	137.879	0.209	0.160
	50000	1520.404	—	0.566	0.752

Table 3: Computing times for two approaches to the CPP.

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_c 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^+. Figure 5 shows the graph G^+ 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

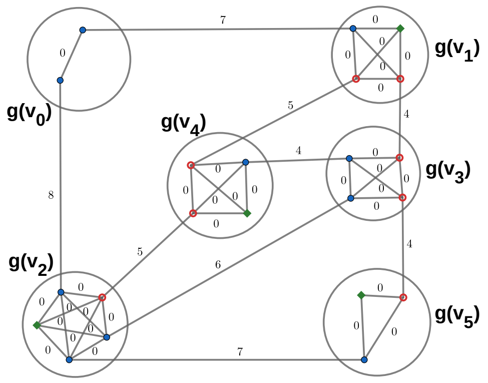


Figure 5: Graph G^+ for the CARP instance in Figure 4.

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 Subsection 2.2). Note that $|g(v)| \geq \deg(v) \geq \deg_R(v) \geq |g'(v)|$ for every $v \in V$. Moreover, $|g(v)| - |g'(v)|$ is always even. (Indeed, $\deg_R(v)$ 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 $D_1, D_2, \cdots, D_{2K}$. 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 $p = \sum_{i=1}^{s} r_i - 2K$. If $p > 0$, we have p R-clones of v_s 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_s.

Figure 6 shows the graph G^{++} for the same CARP instance as before. Since $K = 2$, there are 4 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

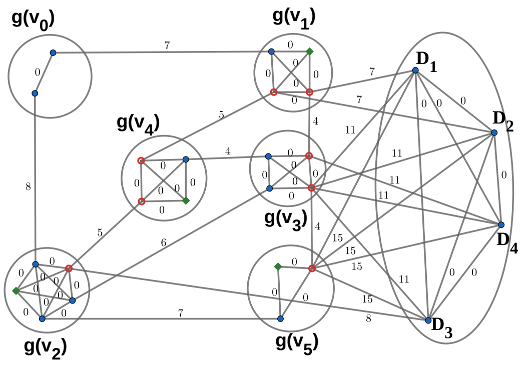


Figure 6: Graph G^{++} for the same CARP instance

to the R-clone of v_2, and the other is connected to an R-clone of v_3. We also have $V_O \setminus V_c = \{v_5\}$. Thus, all dummy nodes are connected to a representative of v_5. 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_s.*

Proof. Let M be a perfect matching in $\tilde{G}$ that matches k dummy nodes with R-clones of v_s, 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_s. Let j_1 and j_2 be two R-clones of v_s 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 $V_c \setminus \{v_s\}$ 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 $\{j_1, j_2\}$, $\{D_1, i_1\}$ 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_s.

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

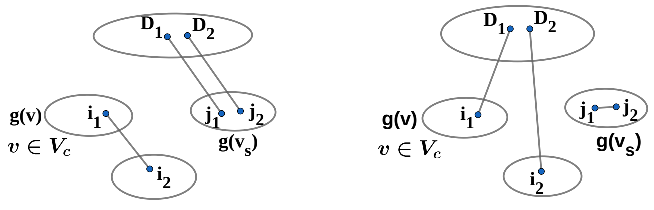


Figure 7: A matching M (left) and a matching M' (right)

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 $\tilde{G}$. 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 $2K - \sum_{i=1}^{s-1} r_i$ R-clones of v_s 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. $\square$

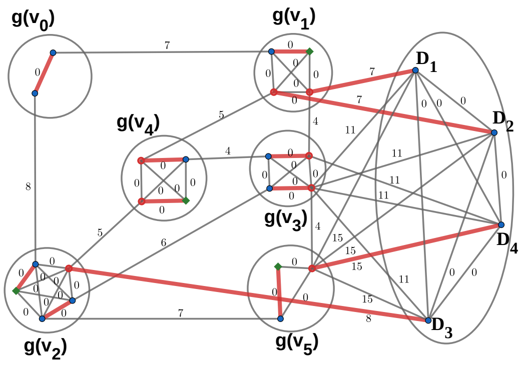


Figure 8: Graph G^{++} and an optimal matching

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 (K) realistic, we set the vehicle capacity (Q) in such a way that K is $|E_R|/500$, rounded up to the nearest integer. The cost of traversing each edge is set to be the length of the corresponding road, in metres, rounded up to the nearest integer. The data for each instance is made at the Lancaster University Data Repository as well.

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 pro-

	Original Graph				LB1		Our LB	
City	n	$\lvert E_R \rvert$	K	Q	$\tilde{n}$	$\tilde{m}$	$\tilde{n}$	$\tilde{m}$
London	1000	1024	3	2708	498	123753	4524	14157
	2000	2141	5	2777	1044	544446	9462	34670
	5000	5594	12	2757	2522	3178981	24906	127586
	10000	11461	23	2760	5150	13258675	50728	374675
	20000	23327	47	2747	10186	51872205	104058	1240173
	50000	59276	119	2757	25628	328384378	262116	6758031
Moscow	1000	986	2	2717	506	127765	4244	11587
	2000	1870	4	2762	1046	546535	8694	28440
	5000	5133	11	2738	2534	3209311	22790	110935
	10000	10518	22	2742	5080	12900660	46520	337427
	20000	21220	43	2758	10052	50516326	94958	1098936
	50000	57601	116	2747	25382	322110271	256068	6528654
Paris	1000	959	2	2735	500	124750	4490	12806
	2000	1940	4	2760	1010	509545	8876	29212
	5000	4908	10	2754	2552	3255076	22016	102164
	10000	9884	20	2760	5108	13043278	44530	307497
	20000	20015	41	2758	10144	51445296	91138	1043119
	50000	52712	106	2746	25350	321298575	236874	5905450

Table 4: Effect of transformation on graph size.

cedure, 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.

		LB1			Our LB		
City	n	T1	T2	Result	T1	T2	Result
London	1000	0.19	0.085	19802	0.015	0.007	19802
	2000	0.895	0.549	41883	0.023	0.021	41753
	5000	6.866	3.787	109887	0.05	0.11	109887
	10000	39.557	28.556	241669	0.182	0.762	241669
	20000	196.482	190.093	507085	0.486	5.231	507085
	50000	930.762	—	—	3.46	293.399	1479122
Moscow	1000	0.268	0.085	37010	0.016	0.005	37010
	2000	1.819	0.542	73604	0.037	0.016	73604
	5000	13.106	3.987	224397	0.077	0.1	224397
	10000	50.759	30.401	546161	0.145	0.856	546161
	20000	207.967	163.09	1254144	0.552	6.312	1254144
	50000	800.54	—	—	5.684	72.698	3929460
Paris	1000	0.194	0.073	23259	0.016	0.005	23259
	2000	1.011	0.49	48370	0.024	0.022	48370
	5000	7.318	4.916	118381	0.068	0.123	118381
	10000	32.711	21.152	250189	0.167	0.69	250189
	20000	145.326	201.194	539548	0.447	3.415	538600
	50000	771.473	—	—	2.928	87.328	1523393

Table 5: Computing times and bounds for two approaches to the CARP.

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.

References

[1] D. Ahr. *Contributions to Multiple Postmen Problems.* PhD 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 R. Möhring and R. Raman, editors, *Proceedings of ESA '02*, pages 64–74. Springer, 2002.

[3] D. Ahr and G. Reinelt. The capacitated arc routing problem: combinatorial lower bounds. In Á. Corberán and G. Laporte, editors, *Arc*

Routing: Problems, Methods, and Applications, pages 159–181. SIAM, Philadelphia, PA, 2014.

[4] F. Barahona. On the computational complexity of Ising spin glass models. *Journal of Physics A*, 15:3241–3253, 1982.

[5] F. Barahona. Planar multicommodity flows, max cut, and the Chinese postman problem. In W. Cook and P.D. Seymour, editors, *Polyhedral Combinatorics*, pages 189–202. AMS, Providence, RI, 1990.

[6] J.M. Belenguer, E. Benavent, P. Lacomme, and C. Prins. Lower and upper bounds for the mixed capacitated arc routing problem. *Computers & Operations Research*, 33:3363–3383, 2006.

[7] E. Benavent, V. Campos, A. Corberan, and E. Mota. The capacitated arc routing problem: lower bounds. *Networks*, 22:669–690, 1992.

[8] P. Berman, A. Kahng, D. Vidhani, and A. Zelikovsky. The T-join problem in sparse graphs: applications to phase assignment problem in VLSI mask layout. In F. Dehne, A. Gupta, J.-R. Sack, and R. Tamassia, editors, *Proceedings of WADS '99*, pages 25–36. Springer, 1999.

[9] G. Boeing. Osmnx: new methods for acquiring, constructing, analyzing, and visualizing complex street networks. *Computers, Environment and Urban Systems*, 65:126–139, 2017.

[10] Á. Corberán and G. Laporte, editors. *Arc Routing: Problems, Methods, and Applications.* SIAM, Philadelphia, PA, 2014.

[11] D. Delling, I. Goldberg, A.V .and Razenshteyn, and R.F. Werneck. Graph partitioning with natural cuts. In *Proceedings of IPDPS '11*, pages 1135–1146. IEEE, 2011.

[12] R. Duan, S. Pettie, and H.-H. Su. Scaling algorithms for weighted matching in general graphs. *ACM Transactions on Algorithms*, 14, 2018.

[13] J. Edmonds. Maximum matching and a polyhedron with 0, 1-vertices. *Journal of Research of the National Bureau of Standards B*, 69:125–130, 1965.

[14] J. Edmonds and E. Johnson. Matching, Euler tours and the Chinese postman. *Mathematical Programming*, 5:88–124, 1973.

[15] R.W. Eglese. Routeing winter gritting vehicles. *Discrete Applied Mathematics*, 48:231–244, 1994.

[16] R.W. Eglese and L.Y. Li. A tabu search based heuristic for arc routing with a capacity constraint and time deadline. In I.H. Osman and J.P. Kelly, editors, *Meta-Heuristics: Theory and Applications*, pages 633–649. Kluwer, Boston, MA, 1996.

[17] M.L. Fredman and R.E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34:596–615, 1987.

[18] H.N. Gabow. Data structures for weighted matching and nearest common ancestors with linking. In D. Johnson, editor, *Proceedings of SODA '90*, page 434–443. SIAM, 1990.

[19] H.N. Gabow. Data structures for weighted matching and extensions to b-matching and f-factors. *ACM Transactions on Algorithms*, 14, 2018.

[20] A.H.M. Gerards. Matching. In M.O. Ball, T.L. Magnanti, C.L. Monma, and G.L. Nemhauser, editors, *Network Models*, pages 135–224. North Holland, Amsterdam, 1995.

[21] G. Ghiani, G. Improta, and G. Laporte. The capacitated arc routing problem with intermediate facilities. *Networks*, 37:134–143, 2001.

[22] B.L. Golden and R.T. Wong. Capacitated arc routing problems. *Networks*, 11:305–315, 1981.

[23] M.-G. Guan. Graphic programming using odd or even points. *Chinese Mathematics*, 1:273–277, 1962.

[24] M.R. Henzinger, P. Klein, S. Rao, and S. Subramanian. Faster shortest-path algorithms for planar graphs. *Journal of Computer and System Sciences*, 55:3–23, 1997.

[25] V. Kolmogorov. Blossom V: a new implementation of a minimum cost perfect matching algorithm. *Mathematical Programming Computation*, 1:43–67, 2009.

[26] B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer, Heidelberg, 6 edition, 2018.

[27] L.Y.O. Li. *Vehicle Routeing for Winter Gritting*. PhD thesis, Department of Management Science, Lancaster University, 1992.

[28] R.J. Lipton and R.E. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36:177–189, 1979.

[29] R.J. Lipton and R.E. Tarjan. Applications of a planar separator theorem. *SIAM Journal on Computing*, 9:615–627, 1980.

[30] L. Lovász and M.D Plummer. *Matching Theory.* AMS, Providence, RI, 2009.

[31] K. Mehlhorn and G. Schäfer. Implementation of $O(nm \log n)$ weighted matchings in general graphs: the power of data structures. *Journal of Experimental Algorithmics*, 7, 2002.

[32] W.L. Pearn. New lower bounds for the capacitated arc routing problem. *Networks*, 18:181–191, 1988.

[33] B. Raghavachari and J. Veerasamy. A 3/2-approximation algorithm for the mixed postman problem. *SIAM Journal on Discrete Mathematics*, 12:425–433, 1999.

[34] B. Raghavachari and J. Veerasamy. Approximation algorithms for the asymmetric postman problem. In R.E. Tarjan and T. Warnow, editors, *Proceedings of SODA '99*, pages 734–741. SIAM, 1999.

[35] Y. Saruwatari, R. Hirabayashi, and N. Nishida. Node duplication lower bounds for the capacitated arc routing problem. *Journal of the Operations Research Society of Japan*, 35:119–133, 1992.

[36] A. Schild and C. Sommer. On balanced separators in road networks. In E. Bampis, editor, *Proceedings of SEA '15*, page 286–297. Springer, 2015.

[37] Z. Win. *Contributions to Routing Problems.* PhD thesis, Institut für Mathematik, Augsburg University, 1987.

[38] S. Wøhlk. New lower bound for the capacitated arc routing problem. *Computers and Operations Research*, 33:3458–3472, 2006.