Improving a Constructive Heuristic for the General Routing Problem

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

To appear in *Networks*

Abstract

The General Routing Problem (GRP) is a fundamental $\mathcal{NP}$-hard vehicle routing problem, first defined by Orloff in 1974. It contains as special cases the Chinese Postman Problem, the Rural Postman Problem, the Graphical TSP and the Steiner TSP. We examine in detail a known constructive heuristic for the GRP, due to Christofides and others. We show how to speed it up, in both theory and practice, while obtaining solutions that are at least as good. Computational results show that, for large instances, our implementation is faster than the original by several orders of magnitude.

Keywords: vehicle routing, arc routing, combinatorial optimisation, heuristics.

1 Introduction

The *General Routing Problem* (GRP) is an $\mathcal{NP}$-hard vehicle routing problem, first defined by Orloff in 1974 [35]. We are given an undirected graph $G = (V, E)$, a cost $c_e \in \mathbb{Q}_+$ for each edge $e \in E$, a set $V_R \subseteq V$ of *required nodes* and a set $E_R \subseteq E$ of *required edges*. The task is to find a minimum-cost closed walk in G that visits each required vertex at least once and traverses each required edge at least once.

The GRP contains several other vehicle routing problems as special cases:

- When $V_R = \emptyset$, we have the *Rural Postman Problem* or RPP [35].
- When $V_R = \emptyset$ and $E_R = E$, we have the *Chinese Postman Problem* or CPP [13, 22].

[*]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

- When $E_R = \emptyset$, we have the *Steiner Travelling Salesman Problem* or STSP [11].
- When $E_R = \emptyset$ and $V_R = V$, we have the *Graphical Travelling Salesman Problem* or GTSP [11].

The RPP is $\mathcal{NP}$-hard in the strong sense [31], and so is the GTSP [11]. This implies that the STSP and GRP are also $\mathcal{NP}$-hard in the strong sense. The CPP, on the other hand, can be solved in polynomial time [13].

The GRP is particularly suitable for modelling vehicle routing problems that involve *road networks* [15, 32, 35]. The edges in E represent roads or road segments, the nodes in V_R represent customers, the nodes in $V \setminus V_R$ represent road junctions, and the edges in E_R represent (usually urban) streets that require service.

Several methods have been proposed for the GRP and its special cases, including both exact methods (e.g., [1, 6–9, 14, 15, 20, 32, 33, 38, 39]) and heuristic methods (e.g., [3, 6, 10, 16, 19, 21, 23–25, 28, 37]). Here, we are interested in a constructive heuristic that is obtained by adapting the well-known Christofides heuristic for the TSP [5] to the GRP. This heuristic has a rather complicated history, and it is difficult to attribute it to any single author (see Subsections 2.3 and 2.4). For brevity, we call it the "C-heuristic".

The goal of this paper is to show how to speed up the C-heuristic, in both theory and practice, without incurring any loss in solution quality. (In fact, the solutions obtained by our version of the C-heuristic are guaranteed to be at least as good as those obtained by the original version.) After that, we present some extensive computational results, on instances created using real road network data. It turns out that, for large instances, our version of the C-heuristic is faster than the original by *several orders of magnitude*.

The paper has a simple structure. Section 2 contains a brief literature review. Section 3 presents the new heuristic, and Section 4 presents the computational results. Section 5 contains some concluding remarks.

Throughout the paper, we assume without loss of generality that end-nodes of required edges do not belong to V_R. We let V_R^+ denote the union of V_R and the set of end-nodes of required edges. Note that $V_R \subseteq V_R^+ \subseteq V$. We sometimes refer to the subgraph $G_R = (V_R^+, E_R)$. The connected components of G_R will be called "R-components". Finally, traversing an edge without servicing it will be called "deadheading".

2 Literature Review

We now briefly review the relevant literature. For purposes of exposition, we cover the CPP, the RPP and the GRP in that order.

2.1 The CPP

To tackle the CPP, one needs to be familiar with matchings and T-joins. Given a graph $G = (V, E)$ with $|V|$ even, a *perfect matching* is a set of edges that touches each vertex exactly once. In the *minimum-weight perfect matching problem* (WPM), one is also given a weight $c_e \in \mathbb{Q}$ for each edge $e \in E$, and one seeks a perfect matching of minimum total weight. Edmonds [12] showed that WPM can be solved in polynomial time. An $O(|V|^3)$ time algorithm was given in [30]. More recently, Gabow [18] presented an $O(|V||E| + |V|^2 \log |V|)$ time algorithm.

Given a graph $G = (V, E)$ 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. In the *minimum-weight T-join problem* (WTJ), one is also given a weight $c_e \in \mathbb{Q}$ for each edge $e \in E$, and one seeks a T-join of minimum total weight. Edmonds and Johnson [13] showed that one can reduce WTJ to WPM as follows. Create a complete graph in which the vertex set is T, and the weight of an edge $\{u, v\} \subset T$ is the cost of the shortest path from u to v in G. Solve the WPM in the complete graph, and replace each edge in the matching with the corresponding shortest path in G. This T-join algorithm takes $O(|V|^3)$ time. A faster algorithm, which runs in $O(|T|(|E| + |V| \log |V|))$ time, was recently found by Gabow [18].

Korte and Vygen [27] gave an alternative reduction from the WTJ to the WPM. The resulting WPM instance is rather large, with $O(|E|)$ nodes and $O(|V||E|)$ edges. If the vertex degrees in G are bounded by a constant, however, the WPM instance has only $O(|V|)$ nodes and edges.

Edmonds and Johnson [13] observed that an optimal CPP solution can be constructed by setting T to the set of nodes incident on an odd number of edges, and then finding a minimum-cost T-join in G. (The edges in the T-join are the ones that need to be traversed twice in the CPP solution.)

Finally, Boyacı *et al.* [4] pointed out that the Korte-Vygen approach is well-suited to CPP instances on road networks, since road networks have bounded degree. Using the Korte-Vygen approach in conjunction with an open-source matching routine, they were able to solve CPP instances with 50,000 nodes in less than a second on a laptop.

2.2 Exact algorithms for the RPP

Christofides *et al.* [6] gave an exact algorithm for the RPP based on Lagrangian relaxation. It could solve instances with up to 180 edges. Corberán and Sanchis [9] presented a cutting-plane algorithm, which could solve instances of around the same size. Ghiani and Laporte [20] presented a full branch-and-cut algorithm, which could solve instances with up to around 600 edges. A different branch-and-cut algorithm was given by Fernández *et* *al.* [14].

The above-mentioned algorithms all begin by transforming the RPP instance into an equivalent RPP instance with $V = V_R^+$. For what follows, we explain this transformation. We take the subgraph $G_R = (V_R^+, E_R)$. For all pairs $\{i, j\} \subset V_R^+$, we add a non-required edge, whose cost is that of the shortest path from i to j in G. We then simplify the resulting graph by deleting all non-required edges (i, j) such that $c_{ij} = c_{ik} + c_{jk}$ for some $k \in V_R^+ \setminus \{i, j\}$. We also delete a non-required edge if there is a parallel required edge with the same cost. We remark that this transformation can increase the number of non-required edges in the RPP instance. In practice, however, it usually decreases it.

Jünger *et al.* [29] show that one can easily transform any RPP instance into a standard TSP instance, as follows. Construct a complete undirected "auxiliary" graph, say G^+, with $2|E_R|$ nodes. Each required edge in G is replaced by a clique of size two in G^+. The cost of an edge in G^+ is set to the cost of the corresponding shortest path in G, with the following exception: the cost of an edge whose end-nodes are in the same clique is set to be the cost of the corresponding required edge in G, minus a large positive number M. Then, a TSP solution in G^+ can be easily converted into an optimal RPP solution. Using this method, in conjunction with their own TSP solver, Jünger *et al.* were able to solve RPP instances with up to 300 edges.

2.3 Heuristics for the RPP

There are also many papers on heuristics for the RPP. Frederickson [16] stated that the famous "spanning tree + matching" heuristic for the TSP, due to Christofides [5], could be modified to give a 3/2-approximation algorithm for the RPP. He did not however give any details. A detailed "spanning tree + matching" heuristic for the RPP was given in Christofides *et al.* [6], and it was proven to be a 3/2-approximation algorithm by Benavent *et al.* [3].

For what follows, we describe the heuristic in [6] in detail. The first step is to transform the RPP instance into an equivalent RPP instance with $V = V_R^+$, as described above. We let G' denote the transformed graph. The next step is to construct a "shrunk graph" as follows. We take a copy of G', shrink each R-component down to a single node, and delete any loops. Then, for each set of parallel edges (if any), we delete all apart from the one with the smallest cost. We then compute a Minimum-Weight Spanning Tree (MST) in the shrunk graph. Let F denote the set of edges in the tree. By construction, $E_R \cup F$ induces a connected spanning subgraph of G'. Finally, we set T equal to the set of nodes in V_R^+ that are incident on an odd number of edges in $E_R \cup F$, and solve the WTJ in G'. The desired RPP solution is obtained by traversing the edges in the T-join and the edges in $E_R \cup F$.

Modified versions of the above heuristic can be found in [24, 37]. Other heuristics for the RPP can be found in, e.g., [10, 19, 21, 23, 37]. For the sake of brevity, we do not go into details.

2.4 The GRP

The GRP has received much less attention than the RPP. We are aware of only two papers on exact algorithms. Corberán *et al.* [7] gave a cutting-plane algorithm, which solved instances with up to 300 edges. Later on, Corberán *et al.* [8] presented a very effective branch-and-cut algorithm, which can solve instances with up to 3000 edges.

There are also very few papers on heuristics for the GRP. Heuristics based on local search algorithms are given in [33, 36]. Jansen [28] extended the RPP heuristic in [6] to a variant of the GRP, in which nodes in V_R must be visited *exactly* once rather than at least once.

Finally, for completeness, we mention that there are several papers on exact algorithms for the STSP (e.g., [1, 15, 32, 38, 39]), plus a paper on a heuristic [25]. The algorithm in [1] is capable of solving instances with over 2000 required nodes.

3 Improving the Heuristic

Observe that the constructive heuristic of Christofides *et al.* [6], mentioned in Subsection 2.3, can be easily extended from the RPP to the GRP. The only alterations are that (i) each node in V_R needs to be treated as a (trivial) R-component in itself, and (ii) those R-components do not need to be "shrunk". (Recall that we are using the convention that end-points of required edges do not lie in V_R.) We will call the resulting heuristic for the GRP the "C-heuristic".

The purpose of this section is to show that the C-heuristic can be significantly improved. In Subsection 3.1, we point out two drawbacks of the original C-heuristic. The improved C-heuristic is described in Subsection 3.2 and analysed in Subsection 3.3.

3.1 Drawbacks of the C-heuristic

In our preliminary experiments with the C-heuristic, we soon noticed two significant drawbacks. The first is that the heuristic can take a very long time when applied to large-scale GRP instances. The second is that the heuristic can yield solutions that are "obviously bad", in the sense that they can be easily improved by visual inspection.

To understand the first drawback better, let us analyse the running time of the C-heuristic. The first step is to compute shortest paths in G between all pairs of nodes in V_R^+. This can be done by calling Dijkstra's single-source shortest-path algorithm $O(|V|)$ times. If we use the Fibonacci heap version of Dijkstra's algorithm [17], each Dijkstra call takes $O(|E|+|V|\log|V|)$ time. Thus, the shortest-path phase takes $O(|V|\,(|E|+|V|\log|V|))$ time.

The next step is to construct the graph G'. The most time-consuming part of that is the deletion of non-required edges (i,j) such that $c_{ij} = c_{ik}+c_{jk}$ for some $k \in V_R^+ \setminus \{i,j\}$. This takes $O(|V|^3)$ time. Compared to that, the time taken to compute the MST in G' is negligible. As for the WTJ phase, it takes $O(|V|^3)$ time using the Edmonds-Johnson approach. Thus, the C-heuristic as a whole takes $O(|V|^3)$ time. This is rather slow when one is dealing with large-scale GRP instances.

The other drawback of the C-heuristic is best illustrated through a couple of examples. First consider the STSP instance in Figure 1(a). Required and non-required nodes are represented by big and small circles, respectively. (In the online version of the paper, the big circles are red.) The costs are also indicated on the edges. Figure 1(b) shows the transformed instance, defined on the graph G'. Note that, for this instance, G' is equal to the shrunk graph. Figure 1(c) shows the shrunk graph, with the edges in F indicated by thick black lines. One can check that the edges $\{1,5\}$ and $\{2,4\}$ form an optimal T-join. Figure 1(d) shows the resulting STSP solution in G', and Figure 1(e) shows the corresponding STSP solution in G. This solution is obviously sub-optimal, since the edge $\{1,3\}$ is traversed four times.

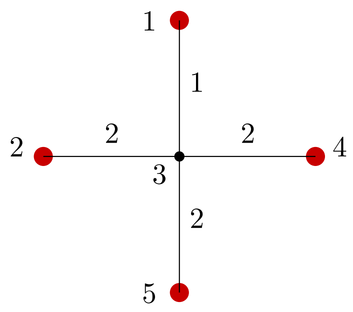
Now consider the GRP instance in Figure 2(a). In this instance, there is only one required edge, represented by a thick red line. Figure 2(b) shows the transformed instance, defined on G'. Figure 2(c) shows the shrunk graph, with an optimal spanning tree represented as thick lines. Figure 2(d) shows the graph G' with the edges in the tree as thick lines. One can check that the edges $\{3,6\}$ and $\{6,8\}$ form an optimal T-join. Figure 2(e) shows the GRP solution in G', and Figure 2(f) shows the corresponding GRP solution in G. This solution is obviously sub-optimal, since we can make a saving by deadheading the edge $\{2,3\}$ instead of the edges $\{2,5\}$, $\{5,6\}$ and $\{3,6\}$.

3.2 The improved C-heuristic

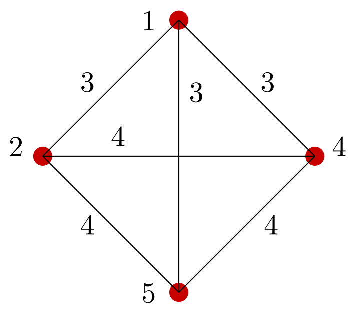
Our improved version of the C-heuristic is intended to overcome both of the above-mentioned drawbacks simultaneously. It has five main phases: a shortest-path phase, a spanning-tree phase, a mapping phase, a sparsification phase, and a T-join phase.

To proceed, we need some more notation. We assume that there are κ R-components, called $C_1,\ldots,C_\kappa$. For $i=1,\ldots,\kappa$, we let $E(i)$ denote the set of required edges (if any) that have both end-nodes in C_i. A path in G that connects a node in C_i to a node in C_j will be called an "(i,j)-link". We let $c(i,j)$ denote the cost of the shortest (i,j)-link in G.

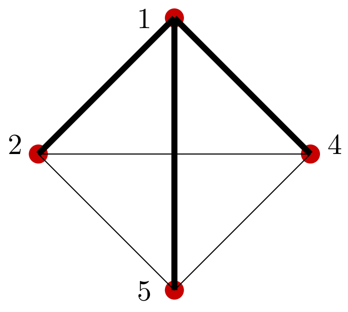
The first phase of our heuristic is described in Algorithm 1. Observe that the algorithm solves only $\kappa-1$ shortest-path problems in G, whereas the standard C-heuristic solves $|V_R^+|-1$ of them. (Note that $\kappa \le |V_R^+|$, with



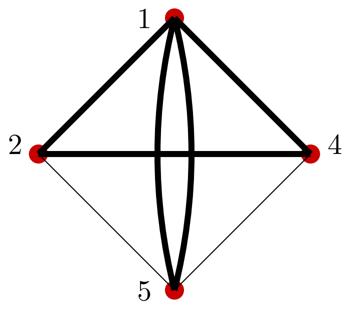
(a) Original STSP instance



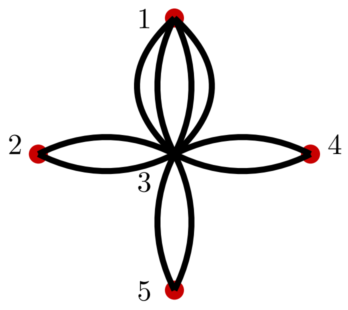
(b) Transformed instance



(c) Shrunk graph and tree

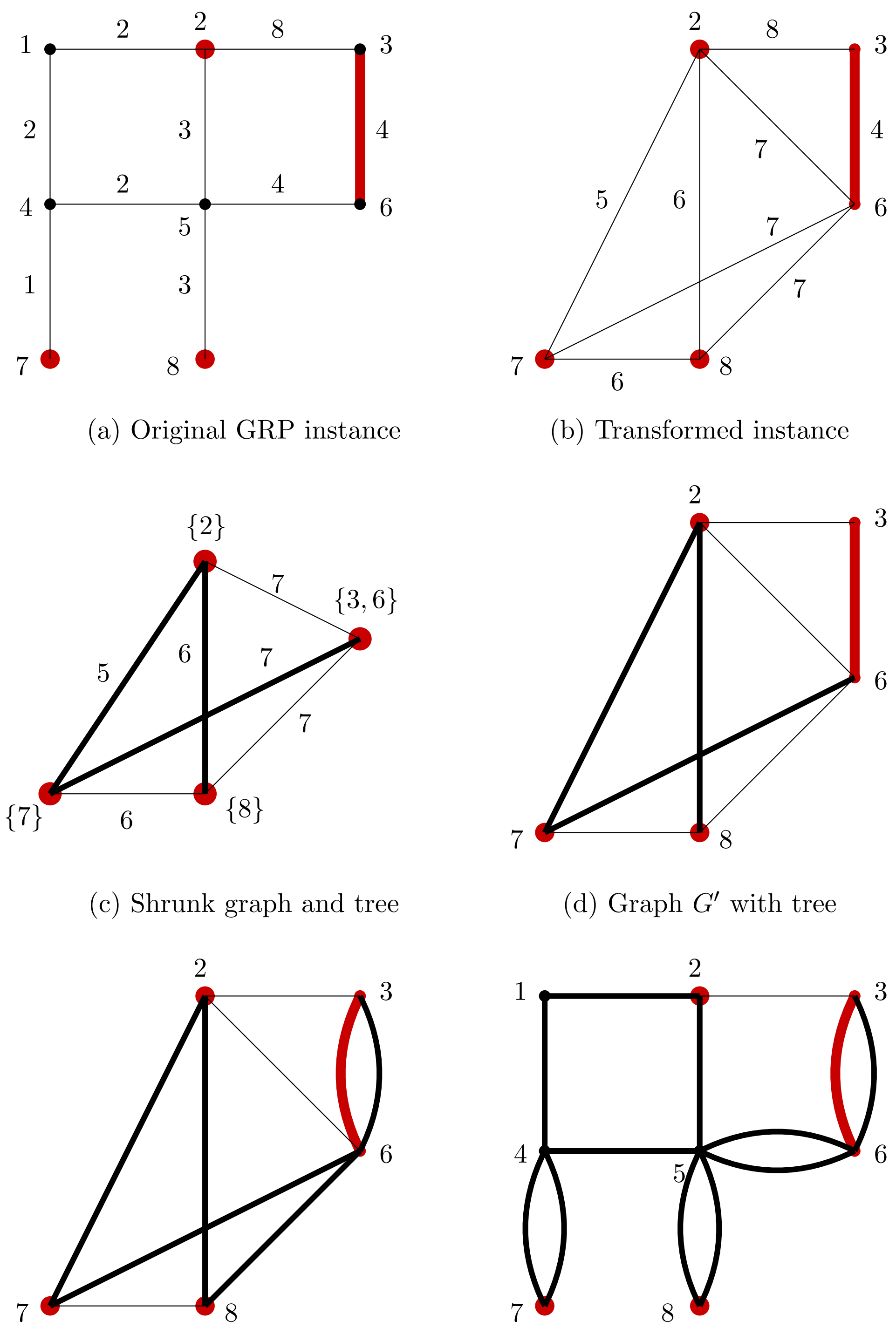


(d) STSP solution in G'



(e) STSP solution in G

Figure 1: Bad STSP instance for the C-heuristic



(a) Original GRP instance (b) Transformed instance

(c) Shrunk graph and tree (d) Graph G' with tree

(e) GRP solution on G' (f) GRP solution on G

Figure 2: Bad GRP instance for the C-heuristic.

equality if and only if $E_R = \emptyset$.) Moreover, we do not bother to construct the graph G'. (It will turn out later on that G' is not actually needed.)

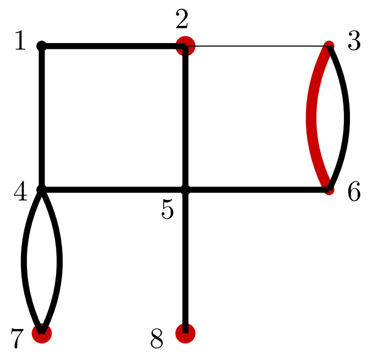
Algorithm 1: Shortest-Path Phase

```
input  : Graph $G$, set of required nodes $V_R \subseteq V$, set of required
         edges $E_R \subseteq E$, edge costs $c_e$
Compute the $R$-components $C_1, \ldots, C_\kappa$ and their edge sets
  $E(1), \ldots, E(\kappa)$;
for $i = 1$ to $\kappa - 1$ do
    Temporarily change to zero the cost of all edges in $E(i)$;
    Let $v$ be an arbitrary node in $C_i$;
    Run Dijkstra's single-source shortest path algorithm to get the
      shortest paths in $G$ from $v$ to all other nodes in $V$;
    Let $T(i)$ be the shortest-path tree;
    for $j = 1$ to $\kappa$ do
        if $j \neq i$ then
            Let $c(i, j)$ be the cost of the shortest path from $v$ to the
              nearest node in $C_j$;
        end
    end
    Restore the costs of the edges in $E(i)$ to their original values;
end
output: Costs $c(i, j)$ and shortest-path trees $T(i)$
```

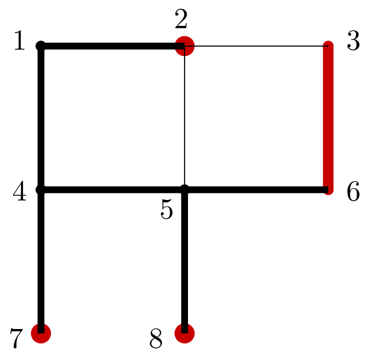
In the second phase, we construct the shrunk graph and compute a MST in it, just as in [6]. Note that the weights of the edges in the shrunk graph are nothing but the $c(i, j)$ values that were stored at the end of phase 1. Note also that the shrunk graph has only κ nodes.

In the third phase, we map the spanning tree onto G. Let F denote the set of edges in the MST. Each edge $\{i, j\} \in F$ corresponds to an (i, j)-link in G, and we can identify the edges in that link using the shortest-path trees that were generated in phase 1. We let F' denote the set of edges in E that correspond to the edges in F. Note that F' is actually a multi-set, i.e., it may contain more than one copy of a given edge. (Indeed, for the example in Figure 1, our set F' will contain three copies of the edge $\{v_1, v_5\}$.)

By construction, the edge set $E_R \cup F'$ induces a connected subgraph of G that contains all required nodes and edges. It may happen, however, that one can drop edges from F' without causing that subgraph to become disconnected. This observation is the motivation for our fourth phase, which we call *sparsification*. Details are given in Algorithm 2.



(a) Graph G with $E_R \cup F'$ in bold



(b) Graph G with $E_R \cup F^-$ in bold

Figure 3: Sparsification applied to the GRP instance in Figure 2

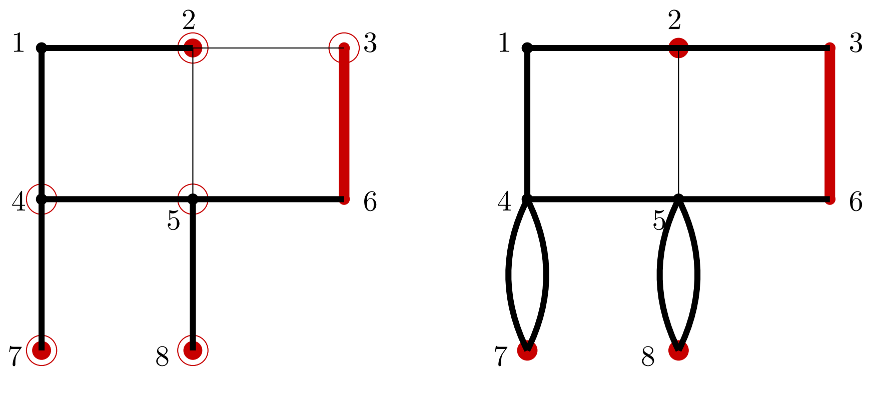
```
Algorithm 2: Sparsification Phase
  input  : Graph $G$, edge costs $c_e$, $R$-components $C_1, \ldots, C_\kappa$, edge
           (multi-)set $F'$
  Construct a graph with vertex set $V$ and edge set $E_R \cup F'$;
  Shrink each $R$-component of the graph to a single node;
  Remove any loops and/or isolated nodes;
  Compute an MST in the resulting graph;
  Let $F^-$ denote the set of edges in the MST;
  output: Edge set $F^- \subset E$
```

Figure 3 shows how the sparsification phase works, when applied to the GRP instance that was presented in Figure 2(a). Figure 3(a) shows the graph G, with the multi-set F' represented by thick black lines. Figure 3(b) shows the same graph with the reduced set F^-. For this instance, sparsification eliminates the edge $\{2, 5\}$, one copy of the edge $\{3, 6\}$, and one copy of the edge $\{4, 7\}$.

Now, consider the graph $G^- = (V, E_R \cup F^-)$. By construction, G^- has one non-trivial connected component, which contains all required nodes and edges. (There may also be some trivial components, consisting of isolated non-required nodes.) In our fifth and final phase, we attempt to make G^- Eulerian at minimal cost.

Let T be the set of nodes in V that are incident on an odd number of edges in $E_R \cup F^-$. To obtain our desired GRP solution, we solve a WTJ problem in G, with the given set T, and add the edges in the T-join to the edges in $E_R \cup F^-$.

Figure 4 shows how the T-join phase works, using the same GRP instance as before. Figure 4(a) is the same as Figure 3, except that the nodes in T are represented by big hollow circles. One can check that the edges $\{2, 3\}$, $\{4, 7\}$ and $\{5, 8\}$ form an optimal T-join. The resulting GRP solution is



(a) Graph G^- and set T (b) Final GRP solution

Figure 4: Illustration of T-join phase for the graph in Figure 2

shown in Figure 4(b).

Note that the GRP solution in Figure 4(b) is cheaper than the one that was found by the traditional C-heuristic, which was already shown in Figure 2(f). One can check that our version of the C-heuristic also yields a cheaper solution for the STSP instance shown in Figure 1(a).

To complete the description of our heuristic, it suffices to explain how we solve the WTJ. We decided to use the Korte-Vygen algorithm rather than the Edmonds-Johnson algorithm. The reason is that most real-life GRP instances are defined on road networks. In road networks, most nodes have degree less than 5. As a result, when applied to a WTJ on a road network, the Korte-Vygen algorithm involves the solution of a matching problem with only $|V|$ nodes and edges (see also [4]).

3.3 Advantages of the modified C-heuristic

We now give some theoretical evidence that the modified C-heuristic is superior to the original. First, we show that, under a mild assumption, the modified version is faster and uses less memory.

Proposition 1. *Suppose the degrees of the nodes in G are bounded from above by a constant (as is the case for GRP instances on road networks). The modified C-heuristic can be implemented so that it runs in $O(|V|^2 \log |V|)$ time and $O(\kappa\,|V|)$ space.*

Proof. Under the stated condition, $|E| = O(|V|)$. Now, Algorithm 1 involves the solution of κ shortest-path problems in G. Using a heap version of Dijkstra's algorithm, this takes $O(\kappa\,|V| \log |V|)$ time and $O(|V|)$ space. Storing the κ shortest-path trees takes $O(\kappa\,|V|)$ space. Constructing the shrunk graph takes only $O(\kappa^2)$ time and space. Computing the MST using

Prim's algorithm takes only $O(\kappa^2 \log \kappa)$ time and $O(\kappa^2)$ space. Mapping the tree F to the multi-set F' takes $O(\kappa\,|V|)$ time and space. If we use Prim's algorithm also in the sparsification phase, it takes $O(|V|^2 \log |V|)$ time and $O(|V|)$ space. Computing the set T takes only $O(|V|)$ time and space. Converting the WPM into a WTJ, as in Korte & Vygen [27], takes only $O(|V|)$ time and space when the node degrees are bounded. Finally, using a decent matching algorithm, such as the one of Gabow [18], the WPM can be solved in $O(|V|^2 \log |V|)$ time and $O(|V|)$ space (since the WPM is defined on a graph with only $O(|V|)$ nodes and edges). □

One can check that the original C-heuristic takes $O(|V|^3)$ time and $O(|V|^2)$ space under the same assumption.

Next, we show that, under mild conditions, the solution found by the modified C-heuristic is at least as good as the one found by the original.

Proposition 2. *If the following two conditions are satisfied, then the modified C-heuristic produces a GRP solution that costs no more than the solution found by the original C-heuristic:*

(a) There is a unique optimal tree F in the shrunk graph.

(b) For each edge $\{i,j\} \in F$, if there is more than one shortest (i,j)-link in G, then all such links connect the same node in C_i to the same node in C_j.

Proof. If condition (a) is satisfied, then both heuristics will produce the same MST F in the shrunk graph. Now, consider the standard C-heuristic. Let F be the set of edges in the spanning tree, let T be the set of nodes used in the T-join phase, and let J be the set of edges in the resulting T-join. Note that $F \cup J$ is a multi-set in general. By replacing each edge in $F \cup J$ with the corresponding shortest path in G, we obtain a (multi-)set $F' \cup J'$ in the original graph G. Note that $F \cup J$ and $F' \cup J'$ have the same cost.

Now consider the modified version of the heuristic. If condition (b) is satisfied, then F' will be the same as in the previous paragraph, and T will be the set of nodes that are incident on an odd number of edges in $E_R \cup F'$. Finally, we let T' denote the set of nodes that are incident on an odd number of edges in $E_R \cup F^-$, and we observe that the union of $F' \setminus F^-$ and J' is a T'-join. In the modified heuristic, however, we compute an *optimal* T'-join. By definition, the cost of F^- together with the cost of the optimal T'-join is no larger than the cost of $F' \cup J'$. □

When the conditions do not hold, the performance of both heuristics can vary depending on the choice of tie-breaking rule. In practice, if we wish to ensure that the modified C-heuristic performs at least as well as the original, it suffices to use the same tree T and the same (i,j)-links in both versions. In other words, given any solution obtained by using the C-heuristic, one can always find a solution, produced by the modified version, which is at least as good. This is also confirmed by the results in the next section.

4 Computational Experiments

In this section, we present the results of some computational experiments. For most of the experiments, we used a PC with an i7-6820HQ processor, running under Windows 10 at 2.7 GHz with 8GB of RAM. For the largest instances, with $10,000$ or $20,000$ nodes, our PC ran into memory difficulties when running the C-heuristic. So, for those instances, we used a workstation with an Intel Xeon W-1390P processor, at 3.5 Ghz and 128 GB of RAM, instead.

All algorithms were implemented with `C#` in the `.NET` framework and compiled with Microsoft Visual Studio 2019. To solve the matching problems, we used the open-source software package `Blossom V` [26]. It runs in $O(|V|^3)$ time in the worst case, but performs extremely well in practice.

4.1 Results for existing benchmark instances

There exist several sets of benchmark GRP, RPP and STSP instances in the literature (see, e.g., [7–9, 20, 23, 25, 32]). Some of them are available on the following web site:

https://www.uv.es/corberan/instancias.htm

These include:

- Three sets of GRP instances taken from [8]. These sets are called *ALBA*, *GRP* and *MADR*.
- Three sets of RPP instances taken from [8]. These sets are called *UR500*, *UR750* and *UR1000*.
- A set of miscellaneous GTSP instances taken from [7].

The results that we obtained for these instances are shown in Table 1. The first three columns of the table show the following for each set of instances: the problem type, the name of the set (where applicable), and the number of instances in the set. The column headed "%gap" shows the average gap between the upper bound from the heuristics and the optimum, expressed as a percentage of the optimum. (For these instances, the improved C-heuristic gave the same upper bound as the classical version.) The remaining columns show the average running time, in seconds, for the original C-heuristic and our version.

It is clear that our version of the heuristic is much faster than the original, especially for the RPP instances. The average percentage gaps are also

Type	Set	#	%gap	C-time	New-time
GRP	ALBA	15	3.03	0.06	0.04
	GRP	10	3.86	0.08	0.04
	MADR	15	3.84	0.08	0.04
RPP	UR500	12	1.91	0.17	0.05
	UR750	12	1.96	0.37	0.07
	UR1000	12	2.29	0.60	0.09
GTSP	—	7	19.30	0.09	0.07

Table 1: Results for several sets of benchmark instances.

encouraging for the GRP and RPP instances. On the other hand, the gaps for the GTSP instances are large. Moreover, it is disappointing that our version of the heuristic gave no improvement in the upper bounds. Closer inspection of the problem data revealed that, for all instances, the transformation from G to G' had already been applied. This meant that F' was equal to F for all instances.

After contacting several authors directly, we managed to find three sets of "raw", untransformed instances:

- 2 manually constructed RPP instances based on the road network of Albaida, a municipality of Valencia, Spain [9].
- 20 STSP instances created using the procedure in [32], which produces random planar graphs that resemble road networks. The procedure takes two parameters: the number of nodes ($|V|$) and the probability that a node is required (p). Further details can be found in [25, 32].
- 10 additional real-world STSP instances created by Interian & Ribeiro [25], based on road and telecommunications networks.

Table 2 shows the results for the two RPP instances. For each instance, we show the number of vertices, the number of edges, the number of required edges, and the number of vertices incident on at least one required edge. We also show the following for each of the two heuristics: the gap between the upper bound and the optimum, expressed as a percentage of the optimum, and the running time, in seconds. It is apparent that these instances are very easy for both heuristics.

Table 3 shows the results that we obtained for the first set of STSP instances. For each instance, we show the probability p, the number of vertices, the number of edges and the number of required vertices, along with the percentage gaps and times, as before. The last column, labelled "%impr", shows the percentage gap improvement gained by using the new heuristic. It is computed as (UB1-UB2)/(UB1-OPT) $\times$ 100%, where UB1

				C-heuristic		New heuristic	
$\lvert V\rvert$	$\lvert E\rvert$	$\lvert E_R\rvert$	$\lvert V_R^+\rvert$	%gap	time	%gap	time
116	174	100	103	0.00	0.48	0.00	0.05
116	174	88	90	0.00	0.13	0.00	0.09

Table 2: Results for 2 RPP instances from [9].

				C-heuristic		New heuristic		
p	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert V_R\rvert$	%gap	time	%gap	time	%impr
	50	69	16	14.07	0.07	14.07	0.06	—
	75	105	25	12.89	0.06	10.96	0.05	14.95
	100	139	33	15.23	0.09	12.40	0.06	18.57
	125	179	41	17.06	0.09	16.01	0.06	6.15
1/3	150	215	50	11.58	0.11	10.32	0.05	10.94
	175	252	58	10.46	0.10	9.67	0.06	7.52
	200	291	66	11.20	0.11	10.73	0.10	4.14
	225	326	75	12.13	0.18	8.50	0.07	29.94
	250	367	83	18.96	0.16	17.42	0.07	8.16
	300	440	100	14.67	0.19	13.93	0.08	4.18
	50	69	33	14.42	0.08	11.66	0.05	19.15
	75	105	50	14.38	0.07	14.38	0.05	—
	100	139	66	19.78	0.09	19.70	0.05	0.42
	125	179	83	16.74	0.10	16.74	0.13	—
2/3	150	215	100	13.98	0.12	13.98	0.06	—
	175	252	116	14.95	0.18	14.83	0.08	0.81
	200	291	133	12.79	0.23	11.60	0.11	9.32
	225	326	150	14.62	0.27	14.52	0.11	0.72
	250	367	166	17.10	0.31	16.95	0.12	0.88
	300	440	200	17.15	0.57	16.49	0.17	3.84

Table 3: Results for 20 STSP instances from [25, 32].

and UB2 are the upper bounds from the old and new heuristics, respectively, and OPT is the optimal value. The improvement is significant, especially on the instances with $p = 1/3$.

Table 4 gives some additional details for the 20 STSP instances. For each instance, we show the probability (p), the number of vertices ($|V|$), the number of edges in the spanning tree in the shrunk graph ($|F|$), the number of edges in G that correspond to the edges in the spanning tree ($|F'|$), and the number of those edges that remain after sparsification ($|F^-|$). We also show the number of additional edges in G that come from the T-join, for the C-heuristic ($|J|$) and the new heuristic ($|J'|$). The benefit of sparsification is apparent.

Finally, Table 5 shows the results for the 10 larger STSP instances from

		Edges from tree			Edges from T-join	
p	$\|V\|$	$\|F\|$	$\|F'\|$	$\|F^-\|$	$\|J\|$	$\|J'\|$
1/3	50	15	22	22	12	12
	75	24	44	42	16	14
	100	32	56	53	25	24
	125	40	74	71	30	31
	150	49	83	78	30	31
	175	57	103	99	29	31
	200	65	117	109	37	42
	225	74	135	124	44	44
	250	82	148	140	48	51
	300	99	182	172	58	62
2/3	50	32	47	44	16	16
	75	49	59	59	24	26
	100	65	78	76	29	29
	125	82	98	98	37	38
	150	99	122	119	46	43
	175	115	138	135	48	47
	200	132	170	163	62	64
	225	149	180	177	75	73
	250	165	195	194	78	77
	300	199	230	226	90	84

Table 4: Additional details for the 20 STSP instances from [25, 32].

			C-heuristic		New heuristic		
$\lvert V\rvert$	$\lvert E\rvert$	$\lvert V_R\rvert$	%gap	time	%gap	time	%impr
1174	1417	58	13.20	0.65	4.40	0.11	66.69
		117	12.09	0.73	5.40	0.17	55.30
		176	9.60	0.82	5.93	0.28	38.25
		234	11.34	1.03	6.08	0.30	46.41
2113	6632	105	6.10	2.49	2.21	0.43	22.83
		211	6.70	2.93	1.92	0.70	16.68
		316	5.35	4.23	2.43	0.82	17.91
3353	8870	167	12.98	5.91	10.02	0.49	63.70
		335	11.34	8.19	9.45	0.83	71.29
		502	13.67	11.39	11.22	1.23	54.66

Table 5: Results for 10 larger STSP instances from [25].

$\lvert V\rvert$	100	200	500	1000
Length	269.9	396.0	645.0	932.7
$\lvert V\rvert$	2000	5000	10000	20000
Length	1323.0	2165.9	3096.5	4366.2

Table 6: Computation of initial squares.

[25]. For these instances, the new heuristic gives significantly better upper bounds than the C-heuristic. Moreover, the new heuristic is more than seven times faster on average.

4.2 New test instances

Since we found few suitable GRP and RPP instances in the literature, we created some of our own. Our instances have $|V| \in \{100, 200, 500, 1000, 2000, 5000, 10000, 20000\}$, and they are based on real road network data for the city of London, extracted using `OpenStreetMap` [34]. For each desired value of $|V|$, we computed the smallest square, centred on Mayfair, that contain the given number of nodes. Table 6 shows the length of the eight resulting squares, in metres.

For each of the eight squares, we created an undirected graph G as follows. We set E to the set of all street segments for which both end-nodes lay in the square. The cost of each edge $e \in E$ was set to the length of the corresponding road, rounded to the nearest metre. We assume for simplicity that all streets are two-way streets.

For each of the eight resulting graphs, we created three GRP instances, by making each node and edge required with probability p, where $p \in \{0.25, 0.5, 0.75\}$. Some relevant statistics for the resulting GRP instances

$\lvert V\rvert$	$\lvert E\rvert$	p	$\lvert V_R\rvert$	$\lvert V_R^+\rvert$	$\lvert E_R\rvert$	κ
		0.25	14	59	29	30
100	124	0.50	10	88	68	23
		0.75	3	97	90	13
		0.25	30	123	61	62
200	252	0.50	17	185	144	46
		0.75	13	193	187	20
		0.25	63	324	162	162
500	656	0.50	47	456	346	118
		0.75	21	494	479	51
		0.25	134	639	328	312
1000	1326	0.50	99	908	684	240
		0.75	48	978	969	93
		0.25	237	1290	676	619
2000	2627	0.50	217	1805	1329	507
		0.75	111	1969	1953	210
		0.25	613	3280	1719	1567
5000	6525	0.50	518	4514	3289	1287
		0.75	287	4921	4927	471
		0.25	1230	6339	3273	3077
10000	12798	0.50	1051	8899	6436	2575
		0.75	538	9823	9618	971
		0.25	1235	11697	6724	4998
20000	26749	0.50	1031	16989	13259	4050
		0.75	443	19179	20019	1154

Table 7: Statistics for the new GRP instances.

can be found in Table 7. We also created 24 RPP instances, simply by taking the GRP instances and setting $V_R = \emptyset$. The data for each instance is made available at the Lancaster University Data Repository [1].

Before running the heuristics, we attempted to find the optimal solution values for the new instances. To do that, we converted all instances into standard TSP instances, using a transformation similar to the one described in [29]. We then fed the resulting TSP instances into `CONCORDE`, the leading open-source exact TSP solver [2]. We set a time limit of one day per instance. We found that `CONCORDE` was able to solve 13 GRP instances and 13 RPP instances within the time limit. For the remaining instances, we ran into serious time and/or memory problems. Table 8 shows the optimal values for the instances that `CONCORDE` was able to solve.

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

$\lvert V \rvert$	p	GRP	RPP
	0.25	4360	3678
100	0.50	6476	5746
	0.75	6937	6815
	0.25	9407	7605
200	0.50	13116	12063
	0.75	14494	13512
	0.25	22929	19844
500	0.50	31825	28746
	0.75	37051	35785
	0.25	42735	36260
1000	0.50	60816	55278
	0.75	73647	71164
2000	0.25	90857	76726

Table 8: Optimal values for some of the new GRP and RPP instances.

4.3 Results for new GRP instances

The next step was to run both heuristics on the 24 new GRP instances. Table 9 shows the results. For each instance, we show the number of nodes $|V|$ and the probability p. Also, for each instance and each of the two heuristics, we show the upper bound ("UB"), the gap between the upper bound and the optimum, expressed as a percentage of the optimum ("%gap"), and the running time, in seconds. We set a time limit of 24 hours for each run. A dash indicates either that the run did not terminate within the time limit, or ran into memory problems.

For the 13 instances that `CONCORDE` was able to solve within the time limit, the average percentage gap was 2.94% for the C-heuristic and 2.75% for our heuristic. Looking at the 21 instances for which the C-heuristic terminated within the time limit, the new heuristic found a better upper bound than the C-heuristic for 13 instances, the same bound for 7 instances, and a worse bound for 1. (It appears that the instance with $|V| = 5000$ and $p = 0.5$ does not satisfy the conditions in Proposition 2.)

Note that the percentage gap decreases dramatically as p increases. A likely explanation for this phenomenon is that the cost of servicing the required edges is effectively a "fixed cost". As p increases, this "fixed cost" makes up a larger proportion of both "Opt" and "UB". (Indeed, when $p = 1$, the GRP reduces to a CPP, and both heuristics will yield the optimal solution.)

In terms of running time, we see that the C-heuristic takes several hours for some instances with $|V| = 10000$. The new heuristic, on the other hand, takes just a few seconds on all instances but the largest. For $|V| \geq 1000$, our heuristic is several orders of magnitude faster than the C-heuristic.

		C-heuristic			New heuristic			
$\|V\|$	p	UB	%gap	time	UB	%gap	time	%impr
100	0.25	4536	4.04	0.09	4472	2.57	0.04	36.36
	0.50	6580	1.61	0.14	6580	1.61	0.04	—
	0.75	6937	0.00	0.14	6937	0.00	0.04	—
200	0.25	9934	5.60	0.19	9930	5.56	0.05	0.76
	0.50	13315	1.52	0.31	13315	1.52	0.05	—
	0.75	14494	0.00	0.29	14494	0.00	0.06	—
500	0.25	24729	7.85	1.54	24639	7.46	0.12	5.00
	0.50	32572	2.35	3.75	32543	2.26	0.11	3.88
	0.75	37210	0.43	4.22	37210	0.43	0.09	—
1000	0.25	45489	6.44	10.21	45365	6.15	0.36	4.50
	0.50	61888	1.76	31.36	61876	1.74	0.28	1.12
	0.75	73911	0.36	41.86	73910	0.36	0.19	0.38
2000	0.25	96605	6.33	102.33	96411	6.11	0.91	3.38
	0.50	127021	—	302.56	126925	—	0.69	—
	0.75	151038	—	385.08	151027	—	0.45	—
5000	0.25	254675	—	1896.34	253351	—	5.70	—
	0.50	328744	—	6061.84	328784	—	4.58	—
	0.75	394331	—	6559.32	394331	—	1.96	—
10000	0.25	515965	—	16667.58	513817	—	14.84	—
	0.50	663559	—	50611.01	663318	—	11.33	—
	0.75	798603	—	71370.72	798539	—	4.41	—
20000	0.25	—	—	—	1317827	—	66.60	—
	0.50	—	—	—	1795515	—	51.83	—
	0.75	—	—	—	2220663	—	20.68	—

Table 9: Results for new GRP instances.

		C-heuristic			New heuristic			
$\|V\|$	p	UB	%gap	time	UB	%gap	time	%impr
	0.25	3696	0.49	0.08	3696	0.49	0.04	—
100	0.50	5850	1.81	0.10	5850	1.81	0.04	—
	0.75	6815	0.00	0.10	6815	0.00	0.04	—
	0.25	8380	10.19	0.11	8380	10.19	0.04	—
200	0.50	12263	1.66	0.23	12263	1.66	0.04	—
	0.75	13512	0.00	0.33	13512	0.00	0.04	—
	0.25	21553	8.61	0.93	21423	7.96	0.09	7.61
500	0.50	29392	2.25	2.49	29385	2.22	0.07	1.08
	0.75	35985	0.56	3.59	35985	0.56	0.06	—
	0.25	38963	7.45	4.77	38904	7.29	0.20	2.18
1000	0.50	56267	1.79	21.93	56267	1.79	0.17	—
	0.75	71318	0.22	33.63	71317	0.21	0.11	0.65
	0.25	82040	6.93	51.64	81900	6.74	0.51	2.63
2000	0.50	112665	—	193.36	112665	—	0.46	—
	0.75	143960	—	317.16	143960	—	0.28	—
	0.25	218046	—	976.69	217281	—	3.15	—
5000	0.50	297873	—	3486.04	297616	—	2.37	—
	0.75	374044	—	5982.47	374044	—	0.91	—
	0.25	437382	—	9409.54	435771	—	7.98	—
10000	0.50	598008	—	33867.66	597815	—	6.04	—
	0.75	759760	—	62245.68	759696	—	2.04	—
	0.25	1221458	—	77533.41	1218154	—	59.73	—
20000	0.50	—	—	—	1711591	—	59.96	—
	0.75	—	—	—	2175634	—	19.09	—

Table 10: Results for new RPP instances.

4.4 Results for new RPP instances

Table 10 presents the results for the 24 new RPP instances. The table has an identical format to Table 10.

The results here are similar to what we saw in the case of the GRP. For the 13 instances that `CONCORDE` was able to solve within the time limit, the average percentage gap was 3.23% for the C-heuristic and 3.15% for our heuristic. Looking at the 22 instances for which the C-heuristic terminated within the time limit, the new heuristic found a better upper for 11 instances and the same bound for 11 instances.

As before, the percentage gap decreases dramatically as p increases, and our heuristic is several orders of magnitude faster than the C-heuristic for $|V| \geq 1000$.

5 Concluding Remarks

The GRP is a classic problem in combinatorial optimisation. We have taken an existing constructive heuristic for the GRP and improved it in terms of both solution quality and running time. Although the improvement in quality is modest in most cases, the improvement in time is dramatic, even amounting to four orders of magnitude in some cases. The improved heuristic works particular well on GRP instances involving road networks.

One obvious potential topic for future research is to improve the heuristic further. We have one suggestion in that regard. A key step in our version of the heuristic is the "sparsification" phase, in which we compute a minimum spanning tree in a suitable shrunk graph. It would be interesting to compute several different spanning trees, rather than just one. This would enable us to generate several heuristic solutions, and then pick the best. Of course, this would come at the cost of increased running time, since several T-join problems would need to be solved.

Acknowledgement: We thank Ángel Corberán for helping us locate some of the benchmark instances. The second author gratefully acknowledges financial support from the EPSRC through the STOR-i Centre for Doctoral Training under grant EP/L015692/1.

References

[1] E. Álvarez-Miranda & M. Sinnl (2019) A note on computational aspects of the Steiner traveling salesman problem. *Inter. Trans. Oper. Res.*, 26, 1396–1401.

[2] D. Applegate, R. Bixby, V. Chvátal & W. Cook (2006) *The Traveling Salesman Problem: A Computational Study.* Princeton NJ: Princeton University Press.

[3] E. Benavent, V. Campos, Á. Corberán & E. Mota (1985) Analisis de heuristicos para el problema del cartero rural. *Trabajos de Estadística y de Investigación Operativa*, 36, 27–38.

[4] B. Boyacı, T.H. Dang & A.N. Letchford (2021) On matchings, T-joins and arc routing in road networks. *Networks*, to appear.

[5] N. Christofides (1976) Worst-case analysis of a new heuristic for the travelling salesman problem. *Technical Report 388*, Graduate School of Industrial Administration, Carnegie Mellon University.

[6] N. Christofides, V. Campos, Á. Corberán & E. Mota (1981) An algorithm for the rural postman problem. *Report IC.OR.81.5*, Imperial College, London.

[7] Á. Corberán, A.N. Letchford & J.M. Sanchis (2001) A cutting plane algorithm for the general routing problem. *Math. Program.*, 90, 291–316.

[8] Á. Corberán, I. Plana & J.M. Sanchis (2007) A branch & cut algorithm for the windy general routing problem and special cases. *Networks*, 49, 245–257.

[9] Á. Corberán & J.M. Sanchis (1994) A polyhedral approach to the rural postman problem. *Eur. J. Oper. Res.*, 79, 95–114.

[10] P.F. de Córdoba, L.G. Raffi & J.M. Sanchis (1998) A heuristic algorithm based on Monte Carlo methods for the rural postman problem. *Comput. Oper. Res.*, 25, 1097–1106.

[11] G. Cornuéjols, J. Fonlupt & D. Naddef (1985) The travelling salesman problem on a graph and some related integer polyhedra. *Math. Program.*, 33, 1–27.

[12] J. Edmonds (1965) Maximum matching and a polyhedron with (0,1) vertices. *J. Res. Nat. Bur. Stand.*, B69, 125–130.

[13] J. Edmonds & E.L. Johnson (1973) Matchings, Euler tours and the Chinese postman problem. *Math. Program.*, 5, 88–124.

[14] E. Fernández, O. Meza, R. Garfinkel & M. Ortega (2003) On the undirected rural postman problem: Tight bounds based on a new formulation. *Oper. Res.*, 51, 281–291.

[15] B. Fleischmann (1985) A cutting plane procedure for the traveling salesman problem on a road network. *Eur. J. Oper. Res.*, 21, 307–317.

[16] G.N. Frederickson (1979) Approximation algorithms for some postman problems. *J. of the ACM*, 26, 538–54.

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

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

[19] G. Ghiani, D. Laganá & R. Musmanno (2006) A constructive heuristic for the undirected rural postman problem. *Comput. Oper. Res.*, 33, 3450–3457.

[20] G. Ghiani & G. Laporte (2000) A branch-and-cut algorithm for the undirected rural postman problem. *Math. Program.*, 87, 467–481.

[21] G. Groves & J.H. Van Vuuren (2005) Efficient heuristics for the rural postman problem. *ORiON*, 21, 33–51.

[22] M.-G. Guan (1962) Graphic programming using odd or even points. *Chinese Math.*, 1, 273-–277.

[23] A. Hertz, G. Laporte & P. Nanchen-Hugo (1999) Improvement procedures for the undirected rural postman problem. *INFORMS J. Comput.*, 11, 53–62.

[24] K. Holmberg (2010) Heuristics for the rural postman problem. *Comput. Oper. Res.*, 37, 981–990.

[25] R. Interian & C.C. Ribeiro (2017) A GRASP heuristic using path-relinking and restarts for the Steiner traveling salesman problem. *Int. Trans. Oper. Res.*, 24, 1307–1323.

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

[27] B. Korte & J. Vygen (2018) *Combinatorial Optimization: Theory and Algorithms* (6th edn). Heidelberg: Springer.

[28] K. Jansen (1992) An approximation algorithm for the general routing problem. *Infor. Process. Lett.*, 41, 333–339.

[29] M. Jünger, G. Reinelt & G. Rinaldi (1995) The traveling salesman problem. In M.O. Ball, T.L. Magnanti, C.L. Monma & G.L. Nemhauser (eds), *Network Models*, pp. 225—330. Amsterdam: North-Holland.

[30] E.L. Lawler (1976) *Combinatorial Optimization: Networks and Matroids*. New York: Holt, Rinehart and Winston,

[31] J.K. Lenstra & A.H.G. Rinnooy Kan (1976) On general routing problems. *Networks*, 6, 273–280.

[32] A.N. Letchford, S.D. Nasiri & D.O. Theis (2013) Compact formulations of the Steiner traveling salesman problem and related problems. *Eur. J. Oper. Res.*, 228, 83–92.

[33] L. Muyldermans, P. Beullens, D. Cattrysse & D.V. Oudheusden (2005) Exploring variants of 2-opt and 3-opt for the general routing problem. *Oper. Res.*, 53, 982–995.

[34] `OpenStreetMap` available at http://wiki.openstreetmap.org

[35] C.S. Orloff (1974) A fundamental problem in vehicle routing. *Networks*, 4, 35–64.

[36] C. Orloff & D. Caprera (1976) Reduction and solution of large scale vehicle routing problems. *Transp. Sci.*, 10, 361–373.

[37] W.L. Pearn & T.C. Wu (1995) Algorithms for the rural postman problem. *Comput. Oper. Res.*, 22, 819–828.

[38] J. Rodríguez–Pereira, E. Fernández, G. Laporte, E. Benavent & A. Martínez–Sykora (2019) The Steiner traveling salesman problem and its extensions. *Eur. J. Oper. Res.*, 278, 615–628.

[39] Y. Xia, M. Zhu, Q. Gu, L. Zhang & X. Li (2016) Toward solving the Steiner travelling salesman problem on urban road maps using the branch decomposition of graphs. *Infor. Sci.*, 374, 164–178.