



# An adaptive scaled network for public transport route optimisation

Philipp Heyken Soares<sup>1</sup> · Christine L. Mumford<sup>2</sup> · Kwabena Amponsah<sup>1</sup> · Yong Mao<sup>1</sup>

Accepted: 3 June 2019  
© The Author(s) 2019

## Abstract

We introduce an adaptive network for public transport route optimisation by scaling down the available street network to a level where optimisation methods such as genetic algorithms can be applied. Our scaling is adapted to preserve the characteristics of the street network. The methodology is applied to the urban area of Nottingham, UK, to generate a new benchmark dataset for bus route optimisation studies. All travel time and demand data as well as information of permitted start and end points of routes, are derived from openly available data. The scaled network is tested with the application of a genetic algorithm adapted for restricted route start and end points. The results are compared with the real-world bus routes.

**Keywords** Public transport · Route optimisation · Network design · Benchmark instance · Genetic algorithm

## 1 Introduction

In the majority of cities around the world, public transport networks have been developed using a combination of local knowledge, planning experience and published guidelines. Most often these networks have evolved over time rather than being designed as a whole (Mumford 2013). Multiple reports have pointed out

---

**Electronic supplementary material** The online version of this article (<https://doi.org/10.1007/s12469-019-00208-x>) contains supplementary material, which is available to authorized users.

---

✉ Philipp Heyken Soares  
[philipp.heyken@nottingham.ac.uk](mailto:philipp.heyken@nottingham.ac.uk)  
Christine L. Mumford  
[MumfordCL@cardiff.ac.uk](mailto:MumfordCL@cardiff.ac.uk)

<sup>1</sup> Laboratory of Urban Complexity and Sustainability, University of Nottingham, Nottingham, UK

<sup>2</sup> School of Computer Science and Informatics, Cardiff University, Cardiff, UK

the insufficiencies of this process and the need for a systematic computer-based approach (see e.g. Zhao and Gan 2003; Nielsen et al. 2005).

The task to generate efficient public transport networks can be treated as five interconnected phases (Ceder and Wilson 1986): (1) route design, (2) vehicle frequency setting, (3) timetabling, (4) vehicle scheduling and (5) crew scheduling. Solving all five phases simultaneously would be optimal. However, due to the high complexity of the task, most researchers focus their efforts on simplified versions of the problem. One common simplification is to focus on the route design phase under the assumption of a fixed transfer time between different routes. This approach will also be used in this paper.

Research for public transport route optimisation requires information on the available transport network and the travel demand. These combined datasets are usually referred to as instances. Many researchers have tested their algorithms on the few fully published instances. A prominent example is the instance published by Mandl (1979) [used e.g. in Ahmed et al. (2019), Arbex and da Cunha (2015) Baaj and Mahmassani (1991), Fan and Mumford (2010) and Nayeem et al. (2014)]. Another often used test instance [e.g. used in Poorzahedy and Safari (2011) and Soehodo and Koshi (1999)], a 24-node network published by Leblanc (1975), is based on the city of Sioux Falls, USA. As these two instances are rather small (15 and 24 nodes), Mumford published four larger instances ranging from 30 to 127 nodes, based on the Chinese city Yubei and the two UK cities of Brighton and Cardiff (Mumford 2013). These have been used by several studies since (e.g. John et al. 2014; Nayeem et al. 2014). In addition to these studies, other researchers built their own test instances based on data from urban areas around the world. Among these are Silman et al. (Haifa, Israel, 1974) (Silman et al. 1974), van Nes et al. (Groningen, Netherlands, 1988), Pattnaik et al. (Madras, India, 1998) (Pattnaik et al. 1998), Feng et al. (Taoyuan-County, Taiwan, 2011) (Feng et al. 2010), Cipriani et al. (Rome, Italy, 2012) (Cipriani et al. 2012), or Gutiérrez-Jarpa et al. (Concepción, Chile, 2017) (Gutiérrez-Jarpa et al. 2017).

Few publications describe the rules of instance generation in detail. One exception is the work by Mauttone and Urquhart (2009) who generated a network with 84 nodes to represent the city of Rivera, Uruguay. The nodes were placed on street junctions close to the centres of housing blocks. This method is only suited to cities built in a strict grid pattern. Another node selection algorithm was proposed by Bagloee and Ceder (2011) to generate instances for Winnipeg, Canada, and Chicago, USA. They select every stop point as a node provided it is further than 300 m from another with a higher expected travel demand. A similar method was also used by John (2016) to generate networks for the UK cities of Nottingham and Edinburgh. Here a fixed number of stops was selected randomly while a minimal distance between selected stops was ensured.

The methods from Bagloee and Ceder (2011) and John (2016) are applicable to most urban areas, but both share the same disadvantage: as the locations of the selected stops within the street network are not fully taken into account, the chance that the resulting network does not reflect the real spatial layout of the city is high (John 2016), especially if the number of selected stops is only a fraction of the total number available.

As the layout of the street network is essential for the design of bus routes, it is important that an instance network sufficiently reflects the characteristics of the street network. We therefore propose a network design procedure which scales down the network to a size manageable for meta-heuristic-based optimisations, while at the same time preserving the characteristics of the urban street network. Scaling down an urban street network is desirable principally to restrict the computation times needed for the passenger objective, which is usually the main bottleneck and leads to an increase of the run time with  $f(N^3)$ ,  $N$  being the number of nodes in the network [see Mumford (2013) for a full explanation]. For our modest desktop set up,<sup>1</sup> we determined 500 nodes, which would result in a runtime of about 600 h, to be the upper-most limit for practical work.

The down-scaling of the street network is achieved by devising simple and robust rules applicable to all urban layouts. The procedure further includes the identification of potential terminal nodes, an aspect vital for route design in an urban context.

We will limit this work to instance generation and route network optimisation with restricted start and end points, as it is part of an incremental approach to more realistic public transport network optimisation. Our generation procedure is used to produce an instance for bus route optimisation in the extended urban area of Nottingham, UK. Further, a route initialisation procedure and a modified heuristic route optimisation algorithm, both specialised for work with restricted route start and end points, are applied to the generated instance. The optimisation results are compared to the real-world bus routes.<sup>2</sup>

The main contributions of this paper are as follows:

1. A novel methodology for generating an instance dataset, by systematically scaling down a street network and utilising census data (see Sect. 2). The generated instance will be published online for free use for all researchers.
2. An additional methodology for transforming pre-existing public transport routes to fit the scaled-down street network (see Sect. 4).
3. A multi-objective genetic algorithm modified for restricted route start and end points, to allow direct comparison with the pre-existing public transport routes, showing potential to improve performance (methodology in Sect. 3 and results in Sect. 5).

---

<sup>1</sup> We used a desktop PC with an Intel i5-6500 3.20GHz Quadcore CPU and 8GB RAM.

<sup>2</sup> It should be noted that this work focuses on mono-modal transport, and the Nottingham tram network, with only two lines in 2011, was not taken into account. For the instance generation, the interoperation of the tram could be included in principle with the rules described in Sect. 2. However, the optimisation algorithm (see Sect. 3) would require modifications beyond the scope of this paper to deal with multi-modal optimisation.

## 2 Generating a scaled instance network for bus route optimisation

This section outlines a systematic approach to generate a street network, consisting of node placement, link generation, and the production of the travel time and demand matrices. The methodology is applied to the extended urban area of Nottingham, UK (see Fig. 1), using UK-specific street and census data. However, the same can be applied to areas outside the UK using other data sources. Examples for potential sources are given in the footnotes 4, 5, and 6.

### 2.1 Problem description

An available transport network can be represented as an undirected graph  $G = \{N, E\}$ , with nodes  $N = \{n_1, n_2, \dots, n_{|N|}\}$  representing access and interchange points (see Sect. 2.3.2) and links  $E = \{e_1, e_2, \dots, e_{|E|}\}$  representing the edges (e.g. streets) connecting the nodes (see Sect. 2.4). Given such a graph, a public transport route can be represented as a list of directly connected nodes  $r = [n_f, \dots, n_l]$  and the public transport network as a set of routes  $R = \{r_1, r_2, \dots, r_{|R|}\}$ . It is necessary to ensure that the first and the last node on each route is a designated terminal node  $V = \{v_1, v_2, \dots, v_{|V|}\} \in N$  which allows u-turns (see Sect. 2.5). Travel times  $T$  and travel demand  $D$  are symmetrical matrices as defined below.

In order to allow an optimisation of the route set  $R$  on the graph  $G$ , the travel time and travel demand between the nodes  $N$  need to be given. We do this in form of two symmetrical matrices:

- $T$ : Gives the travel times  $t_{i,j}$  along the connection between the nodes  $n_i$  and  $n_j$ . Travel times between nodes that are not directly connected are set to  $t_{i,j} = \infty$  and self connections are set to  $t_{i,i} = 0$ .
- $D$ : Gives the number of passengers  $d_{i,j}$  travelling from source  $n_i$  to destination  $n_j$ . Travellers staying at one node are not considered ( $d_{i,i} = 0$ ).

### 2.2 Definitions

We first define some parameters:

- Catchment radius  $c$ :  
Defines circular catchment areas around the nodes used to assign travel demand (see Sect. 2.6). For the Nottingham instance we used  $c = 400$  m, a value widely considered as an acceptable walking distance to a bus stop (Ammons 2001).
- Snapping Distance  $s$ :  
Distance within which two or more junctions are snapped together to be represented by one single node in between (see Sect. 2.3.2). The value for

this snapping distance is not critical and has been chosen to be  $s = c \cdot \sin(\frac{\pi}{4})$ , resulting in  $s = 283$  m.

## 2.3 Constructing the network

The first step is to select which streets will be taken into consideration for the network. This is followed by a process to determine the positions of the nodes on the street map.

### 2.3.1 Defining the street network

For the Nottingham application we base our selection of streets on the integrated network layer from 2011<sup>3</sup> generated by the UK Ordnance Survey.<sup>4</sup> We selected all streets classified as “A-”, “B-” or “Minor Road”. Streets classified as “Local Street” were only selected if they fulfil two conditions: (a) bus stop points exist alongside them<sup>5</sup>; (b) they are not parallel to any street classified as “A-”, “B-” or “Minor Road” within a distance  $s$ . One-way streets are only selected if travel in the other direction is possible on streets within  $s$ .

### 2.3.2 Placing nodes

With the streets selected, the positions of the nodes  $N$  need to be determined. In contrast to the instances generated by Bagloee and Ceder (2011) and John (2016), the nodes in our network do not directly represent bus stop points. Instead they usually represent street junctions. The interpretation is not that buses stop at each node but rather that they travel from node to node and stop at the stop points they encounter on the way. The nodes simply allow us to identify the path a bus will take.

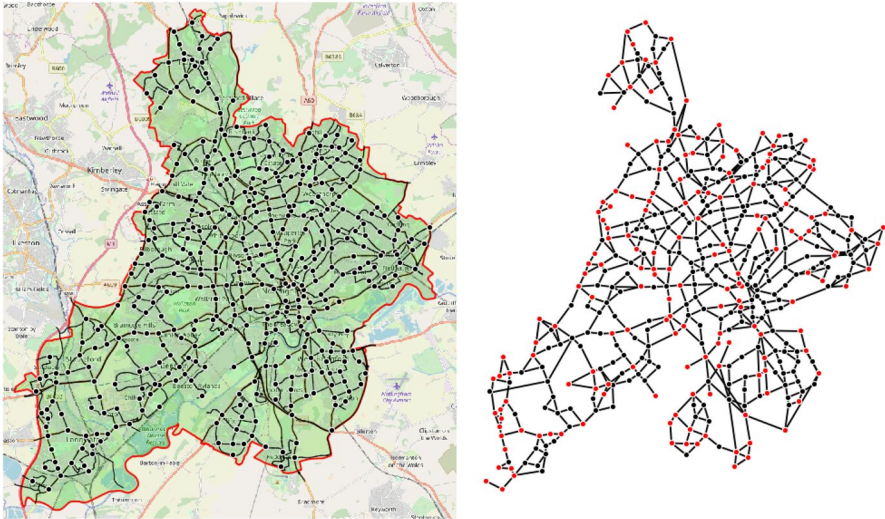
We determine the position of our nodes in three steps:

1. Initial nodes are placed on all points where two or more of the previously selected streets meet. This includes junctions, (t-)intersections, roundabouts, etc.

<sup>3</sup> The same year of census data used to assign the demand (see Sect. 2.6).

<sup>4</sup> Researchers with UK institutional access can download ITN data from <http://digimap.edina.ac.uk/>. The procedure can also be applied to data from other sources. The only constraints are a sufficient classification to select streets available for bus travel and the ability to convert the data to a network dataset for use in ArcGIS to generating the travel time matrix (see Appendix 1). Such data should be available from most national transport authorities, or local authorities. Alternatively, street data from OpenStreetMap (<https://www.openstreetmap.org>) can be used as long as it is sufficiently accurate for the study area.

<sup>5</sup> The location of bus stops can be extracted from the National Public Transport Access Nodes (NaPTAN) which is included in the National Public Transport Data Repository (NPTDR) downloadable from <https://data.gov.uk/dataset/nptdr>. Outside the UK similar datasets should be available from national transport authorities, local authorities or public transport operators. For operators which use General Transit Feed Specification (GTFS) these datasets can be downloaded from <https://transitfeeds.com/>. Also OpenStreetMap (<https://www.openstreetmap.org>) usually provides bus stop locations with a sufficient accuracy.



**Fig. 1** Left: map of the study area together with the street network and placed nodes (source: UK Ordnance Survey, map source: <https://www.openstreetmap.org>, street data source: UK Ordnance Survey; see footnote 4). Right: graph network showing the connections between the nodes with terminal nodes highlighted in red (colour figure online)

2. Nodes within a distance  $s$  of each other form clusters, and each cluster of nodes is snapped together to form a new node. This leads to a simplification error of up to  $\frac{s}{2}$  for the distance between two nodes, but removes the clustering nodes (see Figs. 2, 3).
3. It can happen that two directly connected nodes are too far apart to assign all the demand from zones along the connection to one of the nodes (see Sect. 2.6). In these cases additional nodes need to be placed in-between the regular nodes to properly capture the demand for a bus route along this route (see Fig. 4). The same situation might occur in case of dead-end streets which are longer than  $c$ .

Snapping clusters of initial nodes together has several implications. It further simplifies the representation of transfers between bus stops located at the same node, and it increases the effective length between some snapped nodes compared to a single pair of the snapped nodes (see Fig. 3). However, it drastically reduces the total number of nodes and thereby the processing time. In our case snapping reduces the number of 497 initial nodes by about one third to 324. The mean distance between an initial node and the snapped node placed instead is 73.8 m. The accumulated simplification error for the travel time estimation is about 1.6% of the total travel time along all network edges. As 104 additional street nodes have to be added to ensure all demand can be assigned, giving a total of  $324 + 104 = 428$  (see Fig. 2), the snapping process is important to keep the total number of nodes sufficiently low. Using an estimation of run time of  $f(N^3)$ , the snapping process reduces the run time for the network with 428 nodes by a factor of 3 when compared to a network with about 600 nodes (initial nodes plus street nodes).

## 2.4 Generating links

After placing the nodes, a travel times matrix  $T$  is required for the optimisation stage. Thus the travel times  $t_{i,j}$  between connected nodes need to be determined. We do this by calculating along the shortest path between two directly connected nodes using ArcGIS. The process takes turning restrictions and traffic calming zones into account. The details are given in Appendix 1. Travel times between nodes that are not directly connected are set to  $t_{i,j} = \infty$  and self connections are set to  $t_{i,i} = 0$ .

## 2.5 Determining terminal nodes

One often overlooked aspect in route optimisation studies is that public transport routes cannot terminate everywhere, as this implies that the vehicles are able to turn around and traverse the route in the opposite direction. Therefore, possible terminal nodes require sufficient infrastructure to perform u-turns. It is vital to take this constraint into account when modelling real-world data in an urban context (see e.g. Amiripour et al. 2014).

While it is possible to check the surrounding area of every node manually for turning possibilities, this process would be very time-consuming. Instead we identified possible terminal nodes based on the journey patterns of real-world buses. These journey patterns can be extracted in the form of lists of traversed stop points from the UK's 2011 National Public Transport Data Repository (NPTDR).<sup>6</sup> The stop point lists are then projected on the generated instance network (this process is described in Sect. 4.2). This allows us to determine the nodes at which real-world journey patterns end, either by leaving the study area or by using an existing turn possibility to start the journey in the reverse direction. There may be further nodes which offer such possibilities and are not used by the existing routes. However, as this method already identified 168 terminal nodes in our Nottingham instance (39% of all nodes), we are confident that this process is sufficient. Four additional nodes had to be added to the list of terminal nodes. These are dead-end nodes and in the algorithm used in this paper can only be included at the beginning or end of a route. For them, manual examination ensured the turning possibilities.

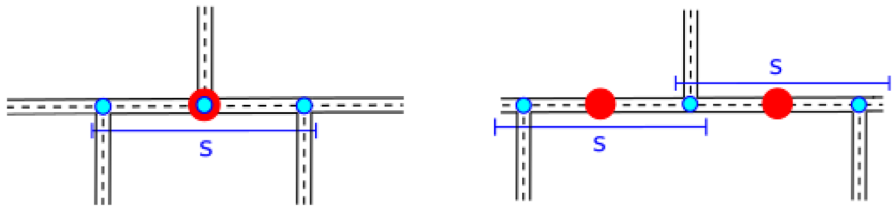
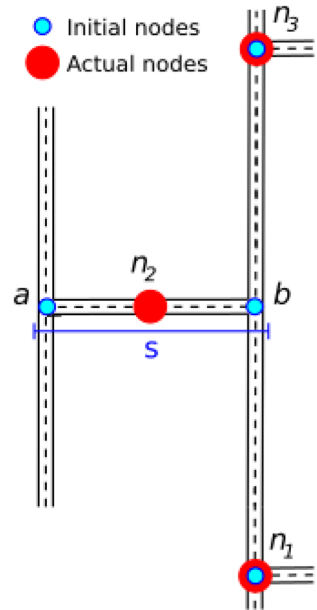
## 2.6 Assigning travel demand

The final step is to generate a node-to-node travel demand matrix  $D$  for the network, which gives the demand  $d_{i,j}$  between the nodes  $n_i$  and  $n_j$ , with ( $d_{i,i} = 0$ ).

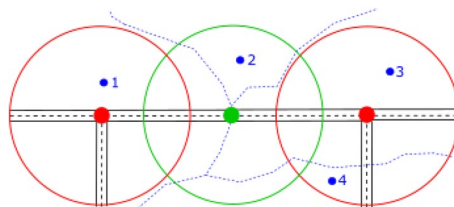
Generating a fully realistic travel demand dataset is a very complex task and not within the scope of this work. What we will show here is not a process for extracting

<sup>6</sup> The 2011 NPTDR contains a snapshot of every public transport journey in Great Britain in a selected week in October 2011 (UK Department for Transport 2015). It can be downloaded from <https://data.gov.uk/dataset/nptdr>. For outside the UK, datasets containing lists of traversed stop points as well as the location of stops should be available from national transport authorities, local authorities or public transport operators. For operators which use GTFS, the datasets can be downloaded from <https://transitfeeds.com/>.

**Fig. 2** Example for snapping nodes in one part of the Nottingham street network: the nodes  $n_1$  and  $n_3$  emerge directly from initial nodes placed on junctions, while the node  $n_2$  results from the snapping together of initial nodes  $a$  and  $b$  (which are closer than distance  $s$ ). It should be noted that node  $n_2$  remains representative of both the junctions it emerged from (i.e.  $a$  and  $b$ ). Thus it is not possible to travel directly between nodes  $n_1$  and  $n_3$  without passing through node  $n_2$ . This generates a maximum simplification error of  $s$  for a travel between node  $n_1$  and node  $n_3$ . In our case the accumulated error is estimated to be 1.6% of the total travel time



**Fig. 3** Left: three initial nodes which are all within snapping distance and which are snapped together to a single node. Right: three initial nodes of which two pairs are within snapping distance. They are snapped into two nodes and the junction between them is represented by either node, depending on the best option for a particular route



**Fig. 4** Example for placing a street node: two regular nodes (red) with catchment area are displayed. The blue dots mark centroids of census zones used to assign the transport demand (see Sect. 2.6). Zone 2 is completely outside the catchment of the two regular nodes; therefore, an additional street node (green) is placed in between them to capture the travel demand for that zone (colour figure online)



travel patterns from raw data sources, but rather a method to transform given travel data between specified zones to a node-to-node demand matrix. Zonal demand data can be generated in different ways. One option is to estimate the number of trips between zones based on the number of origins and destinations in each zone and other parameters [see e.g. Wills (1986) or Wilson (1969)]. This methodology, however, depends much on the quality of the land use data and requires trip survey data or trip counts for calibration. Another methodology well studied in recent years is to extract the travel demand from mobile phone data [see e.g. Golding (2018) or Zhang et al. (2010)], which requires the cooperation of one or several telecommunication companies. A third option is to use survey data. Producing trip surveys is usually time consuming and expensive, and it is preferable to try to access existing survey data if available. In our case we used travel-to-work flow data from the 2011 UK census. It lists the number of commuters from all output areas (in the following origin zones) to all workplace zones (in the following destination zones).<sup>7</sup>

To generate a node-to-node origin-destination matrix (OD-matrix) we take the population weighted centroids of the origin and destination zones<sup>8</sup> and assign them to nodes based on the catchment areas. The catchment area of a node is of circular shape with a radius  $c$  but can be modified under the following circumstances:

- Natural or man-made barriers (rivers, rail tracks, etc.) prevent the commuters from having access to the part of the street network the node represents.
- If a centroid is outside of the catchment area of any node, even when its associated zone overlaps with one or more catchment areas, then these catchments are extended to include this population. (This is to reduce the number of additionally placed street nodes as described in step 3 in Sect. 2.3.2).
- A street node (see step 3 in Sect. 2.3.2) represents the whole street and not just the point where it is placed. (This removes the need to place several nodes along longer streets.)

After this assignment, trips between origin zones and destination zones can easily be converted to trips between the nodes to which the respective zones are assigned. For zones which are within two or more catchment areas, their trips are divided equally between the nodes.

There are of course problems with using travel-to-work data to generate the demand matrix. It represents only a subset of all trips, e.g. trips for shopping or

<sup>7</sup> Output areas and workplace zones are low-level census geographical types. They are redefined for every census. Every output area includes between 40 and 250 households (for the 2011 census on average 309 residents). Workplace zones are created using similar criteria to output areas, to capture employment statistics. It should be noted that output areas and workplace zones overlap (UK Office for National Statistics 2016a). The geometries and flow data can be downloaded from <https://census.ukdataservice.ac.uk/>.

<sup>8</sup> Population weighted centroids are generated by the UK data service, UK Office for National Statistics, together with the respective areas (UK Office for National Statistics 2016b). They can be downloaded from <https://census.ukdataservice.ac.uk/>. If using other datasets where the exact distribution of the population within the zones is not known, the geometric center of a zone can be used instead. These can be calculated with GIS software.

leisure purposes are not included. We will therefore in the following limit our comparisons with the real-world bus routes to the morning rush hour when trips to work dominate the overall travel pattern. This represents a sufficient estimate for this step of our incremental approach. It is worth noting that the demand matrix does not have to be an accurate count of trips but only a weighting for the trip demand within the generated instance network.

### 3 Optimisation procedure

The main goal of this paper is to present a new approach to bus route network design. Thus, we use a genetic algorithm (GA) here simply to produce some viable results to demonstrate the potential of the new network design method. Improvements to the optimisation procedure will follow as future work. For our present purpose, we have implemented an NSGAI algorithm based on the one in John et al. (2014), making some necessary changes to the initialisation process and the genetic operators to accommodate the specified terminal nodes in our Nottingham instances [in the paper by John et al. (2014), any node in the network could act as terminal node for a route].

#### 3.1 Initial definitions

##### 3.1.1 Passenger and operator objectives

Optimisation will attempt to minimise the travel times for individual passengers, while at the same time ensuring the cost to the bus company is reasonable. Our objectives are the average travel time as cost for the passengers, and the total drive time of the route sets as cost for the operator. These are the same objective functions which have been used in John et al. (2014) and other studies (e.g. Mumford 2013). We are using them as it is our aim to only adapt the algorithm described in John et al. (2014) for the use of terminal nodes. Changes to these objectives are possible but beyond the scope of this paper.

The minimum journey time for a given passenger travelling between their origin  $a$  and destination  $b$ , can be given by  $\alpha_{a,b}(R)$ , representing the shortest path in terms of travel time. However, this path is made up of two components: in vehicle travel time and transfer time. In this paper (in line with other recent studies) we will assume that the transfer time is a constant, which is imposed each time a passenger changes vehicle. We set this time constant, representing the average waiting time, to be 5 min.<sup>9</sup> Furthermore, we will assume that each passenger will choose to travel on their shortest-time paths. We define the *passenger objective* for a route set  $R$  to be the mean journey time over all passengers,

<sup>9</sup> This is in line with the definition for “frequent services” given by the UK Department for Transport, which is to have at maximum 10 min between buses (Turfitt 2018). For Nottingham the majority services in the morning rush hour fall into this category.

$$C_p(R) = \frac{\sum_{i,j=1}^{|M|} d_{ij} \alpha_{ij}(R)}{\sum_{i,j=1}^{|M|} d_{ij}} \tag{1}$$

where  $\alpha_{ij}$  is the shortest journey time from  $i$  to  $j$  using route set  $R$ . The operator objective depends on many factors; however, we shall use a simple proxy for operator costs: the sum of the costs (in time) for traversing all the routes in one direction,

$$C_o(R) = \sum_{i=1}^{|R|} \sum_{i=1}^{|r|-1} t_{i,i+1}(r) \tag{2}$$

where  $r$  is a typical route in  $R$ , and  $|R|$  is the number of routes in the route set.  $t_{i,i+1}(r)$  represents the travel time between adjacent nodes in route  $r$ . The passenger cost  $C_p$  (average travel time) and the operator cost  $C_o$  (total route length) will be traded off as conflicting objectives by our multi-objective optimisation algorithm.

Changes in route frequency would effect both costs for the operator and the average travel time, which would in turn require a re-optimisation of the route design. For the present we will set a standard headway of 10 min, in line with UK Government definitions for frequent services (Turfitt 2018).

### 3.1.2 Optimisation constraints

While optimising sets of routes for the two objectives, the optimiser has to follow the specifications of the instance generated in Sect. 2, including the recognition of terminal nodes. These are in addition to the constraints applied in John et al. (2014). The full list of constraints is given below:

1. Each route set  $R$  consists of a predefined number of routes,  $|R|$
2. Each route  $r$  in a route set  $R$  will consist of a number of nodes greater than or equal to  $l_{min}$ , and less than or equal to  $l_{max}$ :  $l_{min} \leq |r| \leq l_{max}$ .
3. No route fully overlaps with another route in the same set.
4. The route set is connected—it is possible to travel between each pair of nodes in the transport network.
5. Each node  $n_i$  is present in at least one route in a route set.
6. No loops or cycles are present in a route.
7. Buses travel back and forth along each route, for which the inbound journey is the reverse of the outbound journey.
8. There is a fixed waiting/transfer time for passengers transferring from one route to another.
9. Each route  $r$  begins and ends at a terminal node  $v \in \{v_1, v_2, \dots, v_{|V|}\}$ , where buses can turn around.

### 3.2 Heuristic construction of route sets

Our initialisation procedure attempts to produce a population of high quality, legal route sets, obeying all of the constraints listed above. We will tackle the construction of route sets in three separate stages:

1. production of a *shortest path usage map* and a *transformed shortest path usage map*
2. generation of candidate routes to form a palette of routes
3. selection of candidate routes to form route sets

#### 3.2.1 First step: constructing a shortest path usage map and transforming it

We use the technique described in Kiliç and Gök (2014) to create a demand map based on edge usage. The process begins by evaluating shortest travel time paths between each pair of nodes for which there is a source to destination travel demand. Then, by recording the total usage of each edge (assuming that each passenger is able to travel along their shortest path) it is an easy matter to create a ‘shortest path usage map’.

Next we perform a simple transformation on the usage map, to reverse the ranks of the labels on the edges. This is where our technique diverges from that of Kiliç and Gök (2014), who convert the usage values directly into edge selection probabilities. In our approach we transform usages into distances, so that the highest usage value becomes the shortest distance and vice versa. We use these transformed values in a deterministic way to create routes, based on shortest path distances through the transformed usage network. The transformation is achieved simply by subtracting the usage on each edge from the total demand for the network as a whole.

#### 3.2.2 Second step: generating candidate routes

This second step produces a palette of routes from which selections can be made to construct the initial population of route sets for our GA. It makes sense to ensure that edges with the highest usage occur in one or more routes within the palette. On the other hand, the inclusion of less busy links will almost inevitably be required to satisfy some of the demand. Thus, although our algorithm begins by selecting the busier edges, the weight on each edge will be very slightly increased every time it is selected, so that the more times an edge is chosen, the less likely it is to be chosen again. We use an arbitrary factor of 1.1 (chosen after some sensitivity tests), making the updated weights equal to 1.1 times the previous weight.

In our initialization we ensure that all routes generated for the route palette begin and end at a terminal node. To this end, our algorithm iterates through all pairs of terminal nodes in turn, choosing the source-destination pair with the highest total demand first, and then the pair with the second-highest total demand, and so on. The algorithm then selects the source-destination node pairs from a pre-sorted list (on total demand for each node pair), created at the start of this second step. For each pair of terminal nodes, the shortest paths from source to destination are then

computed according to the transformed demand usage map. (Note: the edge weights for the shortest paths here are entirely different from those used in stage 1. The stage 1 weights are travel times, but the stage 2 weights are transformed demand usages).

Each shortest path through the transformed usage map forms a candidate route which can be added to our route palette. Following the creation of each new route, the transformed demand usage map is updated by applying the factor 1.1 to the edge weight of each link selected for the route. The iterations cease once we have included each of the nodes in the instance in at least one of the routes included in our palette. It is likely that the algorithm iterates more than once through the list of terminal node pairs, to ensure that all the nodes are included somewhere in the palette. One further point is that in the presence of problem constraints, such as limitations to the lengths of the routes, our procedure will discard any routes that do not obey all the constraints. However, the transformed usage map is updated, whatever the outcome. It is worth noting that, if the imposed constraints are too severe, it may not be possible to generate a route set that obeys all of the constraints, so care is needed when setting the parameters for an instance.

The question now arises whether it is possible to construct legal route sets by making selections from the routes generated by our heuristic construction method. To begin with, we delete duplicate routes from our palette of routes. In the next paragraph we describe a simple method to select and aggregate subsets of routes from the above palette of routes to form route sets that obey all the constraints.

### 3.2.3 Third step: forming route sets by combining routes from the palette of candidate routes

We select routes from the palette one at a time until we have generated an initial population, of route sets  $P_0$  (50 for our experiments), each containing exactly  $|R|$  (i.e. 69) routes. For the first route set, the procedure begins by selecting the first route in the palette. We then add further routes in such a way that: (1) the chosen route has at least one node in common with a route already in the selected subset (beginning with just the first route), and (2) the addition of the selected route maximises the proportion of new nodes included, related to the total number of nodes in the candidate routes. At each iteration, the current subset of routes is tested for inclusion of all the nodes present in the instance. Once all the nodes are present, unused routes from the palette are added at random until the first route set contains  $|R|$  routes. To generate the second route set for the initial population, the second route in the palette is the first route to be selected, and the third route set is seeded with the third route in the palette, and so on.

## 3.3 Genetic algorithm

We adopt an NSGAI genetic algorithm to evolve the generated route sets further. NSGAI is an elitist non-dominated sorting genetic algorithm for the optimisation with multiple objectives (Deb et al. 2002). We constructed our implementation of

the NSGAII after the one in John et al. (2014) but made some changes to adapt to the use of terminal nodes. A flow diagram of the algorithm can be seen in Fig. 5.

### 3.3.1 Outline of our implementation of NSGAII

To save space we do not give full details of the genetic algorithm here, but refer the interested reader to Deb et al. (2002) and John et al. (2014). In summary, let  $P_0$  be an initial population with  $|P|$  route sets. All these route sets are evaluated and sorted into a series of Pareto fronts  $(f_1, f_2, \dots)$  based on their level of domination. Each solution is assigned a fitness value according to its front membership, with  $f_1$  being the fittest,  $f_2$  the second fittest, and so on. An offspring population  $Q_0$ , also of size  $|P|$ , is then generated from  $P_0$  through binary tournament selection, crossover and mutation (see Fig. 5).

Next, the combined (mating) population  $M_0 = P_0 \cup Q_0$  is used to select  $|P|$  route sets as a new parent population  $P_1$ . This selection is primarily based on domination, but crowding distance is also taken into account (see Fig. 5). Crowding distance is an additional fitness measure used to obtain a wide spread of solutions that adequately covers the full extent of the Pareto front (Deb et al. 2002).  $P_1$  is then used to generate  $Q_1$  via binary tournaments, crossover and mutation as previously. The stages of NSGAII repeat for a predetermined number of generations.

### 3.3.2 The genetic operators

*Crossover:* In the crossover step, route sets of the current parent population  $P_k$  are selected in binary tournaments and used to generate in total  $|P|$  offspring route sets. For each parent route set it is decided probabilistically<sup>10</sup> if it is either directly inserted in the offspring population, or if it performs a crossover operation with one other parent to generate an offspring route set. In the crossover operation, routes from both parent route sets are selected in alternation to construct the new route set. The route selection prefers routes which visit nodes not already included in the offspring route set. Before the new route set is allowed to enter the offspring population, a feasibility test is applied (see below). If it fails the test, the crossover process is restarted.

*Mutations:* In the mutation stage, all route sets of the offspring population undergo changes through mutation operations. The number of mutations in each route set is defined by a binomial distribution  $B(|R|, \frac{1}{|R|})$ , with  $|R|$  being the number of routes in each route set. For every mutation one of the following mutation operations<sup>11</sup> is selected at random:

<sup>10</sup> The probability to perform a crossover with another route set is in our case set to  $p_{cross} = 0.9$ .

<sup>11</sup> We use in principle the same mutation operators as John et al. (2014). The operators Delete Node and Add Node, both originally from Mumford (2013), had to be modified to work with terminal nodes. The operator Exchange is used as it was originally proposed by Mandl (1979). The operators Merge and Replace, both from John et al. (2014), stay as they were, apart from the changes in the route generation procedure. Only the operator Remove Overlapping (John et al. 2014) is deleted from the set of mutations as it is part of the feasibility test.

- “Delete nodes”: selects a route at random, ensures that it includes more than two terminals and starts to delete nodes from one of its ends until it reaches another terminal. If less than  $Z$  nodes<sup>12</sup> are deleted in this way, another route is chosen and the process is repeated until at least  $Z$  nodes are deleted.
- “Add nodes”: selects a route at random and adds nodes at one of its ends. The new nodes are selected by a guided random walk to ensure that the route ends at the next possible terminal. If fewer than  $Z$  nodes (see footnote 12) are added in this way, another route is chosen, and the process is repeated until at least  $Z$  nodes are added.
- “Exchange”: extracts two intersecting routes at random, splits them at one common vertex and forms two new routes from the split paths.
- “Replace”: calculates which route satisfies the least demand and deletes it. The route is replaced by a newly generated route<sup>13</sup>.
- “Merge”: randomly selects two routes which share one terminal node and do not overlap elsewhere and merge them into one route. Afterwards a new route is generated<sup>13</sup>.

After every mutation operation, the mutated route set is subject to a feasibility test<sup>14</sup> (see below). If it fails the test, all mutations are undone and a new mutation operation is selected at random.

Further details of add nodes and delete nodes are given in the appendix Sects. 1 and 2, respectively. The other operators are changed very little from those in John (2016).

*Feasibility test:* Every offspring route set generated by a crossover as well as every route set changed by mutation is subject to a feasibility test to ensure that all route sets obey the constraints listed in Sect. 3.1.2. However, to avoid rejecting invalid solutions that are easily corrected, we implemented two repair operations to fix two common constraint violations:

- “Add missing nodes”: In cases where nodes are missing, we adopt a repair procedure based on that used in Mumford (2013). However, this method required some adaptation to work effectively with pre-defined terminal nodes and is explained in detail in Appendix 3.
- “Replace overlapping”: In cases where one route fully overlaps with another route in the same set, we replace the shorter route with a newly generated route<sup>13</sup>. Details are given in John (2016).

<sup>12</sup>  $Z \in [0, \frac{n_{max}}{2}]$  is randomly selected at the beginning of the operation [as in Mumford (2013)].

<sup>13</sup> The new route is generated by a version of the generation procedure of Shih and Mahmassani (1994) which is modified for the use of terminal nodes. The original procedure selects the node pair with the highest travel demand that is not yet directly connected and generates a shortest path route between them. For the use with terminal nodes it has to be ensured that only terminal node-pairs can be picked.

<sup>14</sup> It is important to note that for the operation “Delete Nodes” the repair function is disabled as it could undo the mutation.

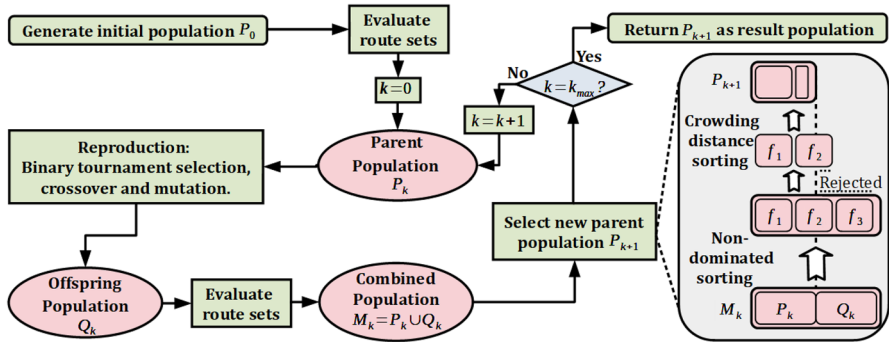


Fig. 5 Flow diagram for NSGAI. In every generation  $k$  the parent population  $P_k$  is used to generate an offspring population  $Q_k$ . The parent population of the next generation  $P_{k+1}$  is then selected from the combined population  $M_k$  based on domination and crowding distance

These repair algorithms are automatically run when the corresponding constraint validation is detected. If the repair is successful, the feasibility test continues for the remaining constraints.

### 4 Comparison of optimisation result and real bus routes

One way to evaluate the effectiveness of our bus routing optimisation is by comparison with real-world bus routes. Such comparisons are relatively rare in the available literature. One reason for this could be that the often used published instances, such as the Mandl instance (Mandl 1979), do not come with a given set of real-world bus routes. From the researchers who create their instances, a few have made comparisons with the real-world routes within their study area [e.g. Bagloee and Ceder (2011), Bielli et al. (2002), or Cipriani et al. (2012)].

#### 4.1 Necessary network reduction

For our Nottingham instance generated with the method in Sect. 2, a comparison between the real-world bus routes and the routes generated by the algorithm described in Sect. 3 is not straightforward. As the evaluation procedure for route sets is based on the average travel time between the different nodes, it consequently requires that all nodes of the network are included in the route sets. The real-world route set, however, does not include all the nodes generated by the rules in Sect. 2.3, as bus operators drop those parts of the network they consider too unprofitable. To make a comparison with the real-world routes possible, we first need to generate a reduced version of the Nottingham instance in which all the nodes not visited by the real-world routes are excluded (see Fig. 6).

Excluding the non-visited nodes essentially generates a second instance network. Travel times between the remaining nodes have to be recalculated and the travel demand has to be reassigned, as described in Sects. 2.4 and 2.6, respectively.



Demand from zones which are not within the catchment area of any of the remaining nodes will not be taken into account.

The final reduced instance has 376 nodes, 52 less than the full instance. The total demand is reduced by 14%. In the following we will run experiments with both instances to point out the differences necessary to serve different network sizes.

## 4.2 Extracting real-world routes

As mentioned in Sect. 2.5, it is possible to extract information about the existing bus routes from the UK's 2011 National Public Transport Data Repository (NPTDR)<sup>6</sup>. The NPTDR provides this information in the form of 990 Journey Patterns (JP). A JP consists of a list of stop points the buses traverse and the starting times of the bus journeys. To convert this information into a route set comparable with the results of our optimisation requires a three-step process: filtering JPs by starting time, converting stop points to nodes, and filtering out overlaps.

### 4.2.1 Filtering journey patterns by starting time

As not all Journey Patterns are in use at a given time, we need to filter out those routes which fit our demand data. As we generated our demand data from travel to work data, we aim to include only JPs active during the morning rush hour into our real-world route set. We therefore select only the 210 JPs which, according to the journey starting times, cover the entire time window of 7:30 am to 10:30 am.

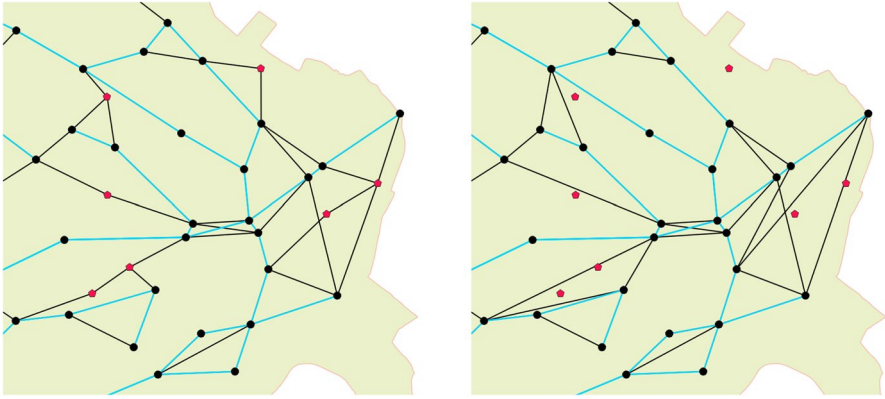
### 4.2.2 Converting stop point lists to node lists

The NPTDR comes with a 2011 version of the National Public Transport Access Nodes (NaPTAN) containing the location of all bus stop points. We use this to link every stop point with the node closest to it, up to a distance  $\rho = \frac{s}{2} + 2s_p$ . Where  $\frac{s}{2}$  is the maximal distance between a node and a junction it represents.  $s_p$  is the usual distance between a junction and a bus stop. For the Nottingham street network, we found  $s_p = 30$  m which results in  $\rho = 202$  m.<sup>15</sup> Mapping the stop point lists of the JP to node lists allows the real-world routes to be compared with the routes generated by the optimisation algorithm.

### 4.2.3 Filtering out overlaps

Journey Patterns always contain only one journey in one direction. In contrast, the routes in our optimisation are undirected and represent travel in both directions. Also, there are several bus routes which consist of similar JPs which are used in alternation. To filter out these overlaps, we add the node lists of the JPs in randomised order to the real-world route set. Every time a new node list  $j$  is added it

<sup>15</sup> It should be noted that some post-processing is always needed to make sure that all stop points are correctly allocated.



**Fig. 6** Illustration of the differences between the original and the reduced instance network in the area of Gedling at the east end of the study area. Left: the real-world bus routes in projection on the original instance network (blue). Nodes of the network are represented as black circles and connections unused by the real-world routes as black lines. Red pentagons mark the nodes which are not served by the real-world routes. Right: real-world bus routes in projection on the reduced network: All nodes not served are removed from the network and the remaining nodes are connected accordingly (colour figure online)

is compared to all routes  $i$  already part of the real-world route set. This is done by calculating  $\omega_{j,i}$  the overlap of  $j$  and  $i$  as well as the length  $\lambda$  of  $\omega$ ,  $j$  and  $i$  :

- If  $\lambda(\omega_{j,i}) = \lambda(j)$ :  $j$  is fully overlapped by  $i$  and is not inserted.
- If  $\lambda(\omega_{j,i}) = \lambda(i)$ :  $i$  is fully overlapped by  $j$  and  $j$  replaces  $i$ .
- If  $|\lambda(\omega_{j,i}) - \lambda(j)| \leq m_i$  and  $|\lambda(\omega_{j,i}) - \lambda(i)| \leq m_i$  and  $|\lambda(i) - \lambda(j)| \leq \frac{m_i + m_j}{2}$  with  $m_x = \min(2, \frac{\lambda(x)}{10})$ :

There are only very small variations between  $j$  and  $i$ . If  $j$ 's first journey starts earlier than  $i$ 's first journey,  $j$  replaces  $i$ . Otherwise  $j$  is not inserted.

After this final filter process, 69 JPs remain as routes in our real-world route set.

## 5 Experimental results

We ran two computer experiments. Experiment 1 used the reduced instance (following Sect. 4) and its results can be directly compared with the real-world route set. Experiment 2 used the full instance (following Sect. 2). As the full instance network includes about 14% more nodes than served by the real-world routes the results of experiment 2 cannot be directly compared. However, optimising the route set on the full instance network is, nevertheless, a useful exercise in demonstrating the conditions under which a public transport service for the entire area can be realised.

In each case, we generated  $|P| = 50$  initial route sets and optimised them with the described GA over 200 generations. Each of the route sets has  $|R| = 69$  routes, the

same number as the real-world route set.<sup>16</sup> For experiment 1 the maximal number of nodes per route was set to  $l_{max} = 45$ , which is around 10% more than the longest real-world route.<sup>17</sup> For experiment 2 we used  $l_{max} = 52$ , to reflect the approximately 14% larger size of the instance network. The minimal number of nodes per route was set to  $l_{min} = 3$  for both experiments, one less than the shortest real-world route.

The results of both experiments are presented in Fig. 7. In both cases the evaluation results of the final route sets (black dots) form a clear Pareto front. In experiment 1 five route sets surpass the performance of real-world route sets (black x) in both objectives. Even for experiment 2 the results show that route sets serving the entire study area can be optimised to achieve evaluation results close to those of the real-world route set in the reduced instance.

To ease the discussion, the results at four key positions in the Pareto fronts for both experiments are highlighted and compared. At the extremes, the most passenger-friendly route sets (red) and the most operator-friendly route set (blue) are identified in the figure. Further, in yellow we can see the most passenger-friendly route set with shorter total route length than the real-world route set, while the most operator-friendly route set with shorter average travel time than the real-world route set is identified in green.

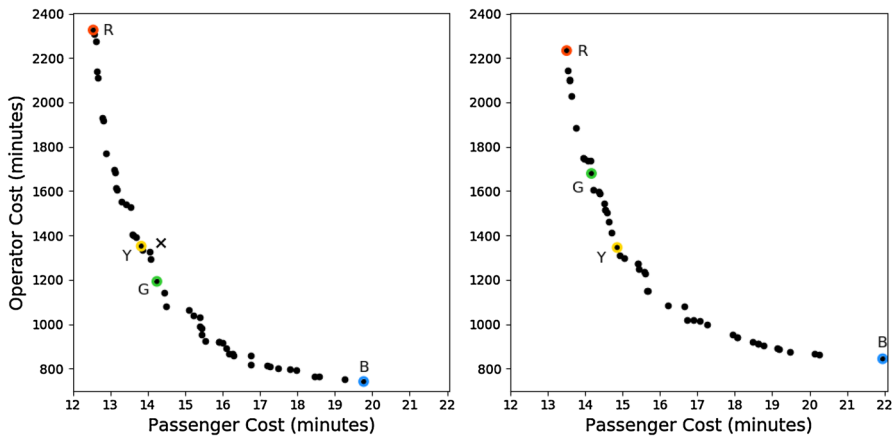
For experiment 1, the highlighted results are shown in more detail in Table 1. From the table we can see that the route set marked in yellow has slightly cheaper operator costs ( $-1.24\%$ ) than the real-world route set, shortening the average travel time by half a minute. On the other hand, the route set marked in green reduces the operator cost by 12.9%, yet still achieves a better passenger cost ( $-0.7\%$ ) than the passenger cost for the real route set. The general trend towards higher optimisation potential on the operator side is also present at the extreme ends of the Pareto fronts. We can observe that the most passenger-friendly route set (red), reduces the average travel time by 12.6% (to 12.5 min) but increases the operator-cost by 69.8%. On the other hand, the most operator-friendly route set (blue) reduces the operator cost by 46% by driving up the average travel time by 38.5% (to almost 20 min).

While the yellow and green marked route sets of experiment 1 indicate that the real-world routes can be improved upon with our optimisation procedure, yellow and green marked route sets of experiment 2 show that it is possible to construct route sets serving the entire study area producing very similar operator and passenger costs to the real-world route set. The green route set offers approximately the same travel time as the real-world routes (14.2 min), although this comes at an increase in operator cost of about 22.7%. The yellow route set serves the larger network for approximately the same operator cost ( $-1.7\%$ ) by prolonging the average travel time to 15 min.

It should be noted that all these comparisons are based on route optimisation only, and thus assume a fixed frequency of 10 min for all routes compared. Our

<sup>16</sup> We have chosen this after a sensitivity analysis showed that changing the number of routes does not lead to a general improvement of results.

<sup>17</sup> The higher limit reflects the possibilities for planners to construct slightly longer routes than currently exist.



**Fig. 7** Evaluations of the final population of 50 route sets from the GA (black dots) for both experiments, the reduced instance (left), and the full instance (right). In each plot four evaluation results are highlighted: the most passenger-friendly route set (red-R), the most operator-friendly route set (blue-B), the most passenger-friendly route set with shorter total route length than the real-world route set (yellow-Y), and the most operator-friendly route set with shorter average travel time than the real-world route set (green-G). Further, passenger and operator cost for the real route set is marked on the reduced instance (left) with an X (colour figure online)

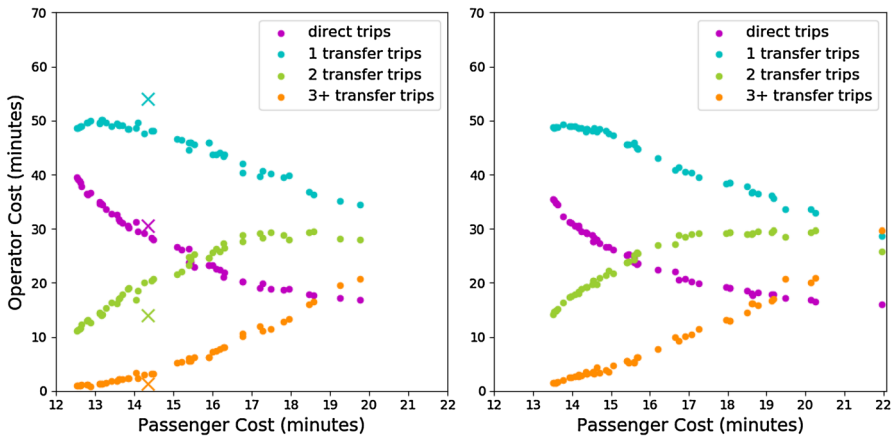
**Table 1** Comparison between optimisation results (as highlighted in Fig. 7) and the real world for optimisation criteria and transfer statistics

	Real routes	Red		Blue	
Average travel time	14.3 min	12.5 min	- 12.6%	19.8 min	+ 38.5%
Total route length	1369 min	2325 min	+ 69.8%	741 min	- 45.9%
% direct trips	30.6%	39.5%	+ 8.9%	16.9%	- 13.7%
% of 1 transfer trips	54.1%	48.5%	- 5.6%	34.4%	- 19.7%
% of 2 transfer trips	13.9%	11.1%	- 2.8%	28.0%	+ 14.1%
% of 3 + transfer trips	1.4%	0.9%	- 0.5%	20.7%	+ 19.3%
	Real routes	Green		Yellow	
Average travel time	14.3 min	14.2 min	- 0.7%	13.8 min	- 3.5%
Total route length	1369 min	1192 min	- 12.9%	1352 min	- 1.24%
% direct trips	30.6%	29.2%	- 1.4%	30.5%	- 0.1%
% of 1 transfer trips	54.1%	47.7%	- 6.4%	48.5%	- 5.6%
% of 2 transfer trips	13.9%	20.0%	+ 6.1%	18.8%	+ 4.9%
% of 3+ transfer trips	1.4%	3.1%	+ 1.7%	2.27%	+ 0.9%

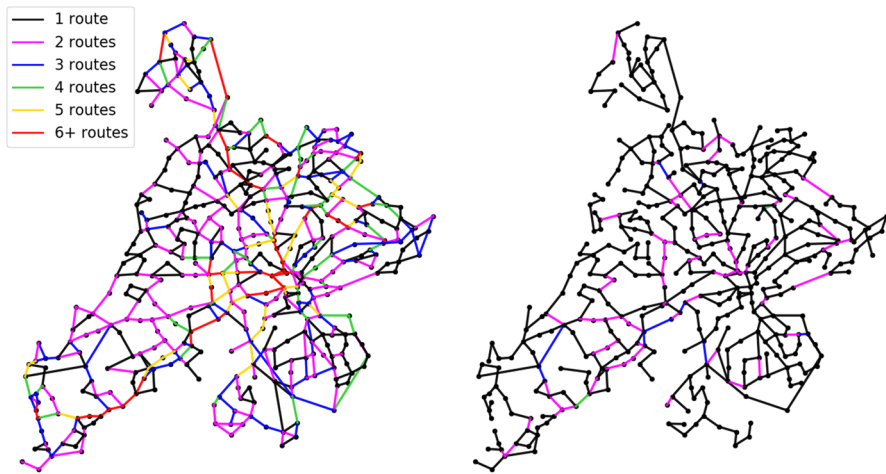
future work looks to improve the realism of these comparisons by using more accurate demand data as well as including aspects such as frequency setting and multi-modal interactions.

Table 1 further shows transfer statistics of the route sets, another performance measure used in the literature (see, e.g. Feng et al. 2010). The transfer statistics show the percentage of travellers reaching their destination, with none, one, two, three or more transfers. For the four marked route sets we see an evident increase in transfers for route sets with shorter total route length and longer average travel times. This observation is confirmed by the transfer statistics for all route sets shown in Fig. 8. This graphic shows a clear correlation between the passenger cost and proportion of passengers needing to make a specific number of transfers, with the number of passengers making zero or one transfer decreasing and the number making two or more transfers increasing, with increasing passenger cost. This behaviour is present in the results of both experiments indicating that it is independent from the specific network or network size. The reason for it lies in the differences in route coverage density in the network as shown in Fig. 9 for the two extreme Pareto route sets for the full instance. By the route coverage density, we simply mean the number of routes covering each link of the network.

Figure 8 further shows that the real-world route set (displayed by ‘X’ markers) has a comparatively low number of transfers, minimising transfers at the expense of operator costs. One potential explanation for this low number of transfers is the fragmentation of the Nottingham bus market. In the absence of tickets valid for all companies, direct travel is monetarily attractive for passengers. Companies optimise their networks individually to attract passengers with direct travel. This leads to a network with higher operator costs overall, due to unutilised transfer potential. However, more research would be required to confirm this.



**Fig. 8** Transfer statistics for the reduced instance experiment (left) and full instance (right). The coloured dots show the percentage of travellers reaching their destination with direct trips (purple), with one transfer (blue), two transfers (green), or three or more transfers (orange). Further, ‘X’ markers display the transfer statistics for the real-world route set (colour figure online)



**Fig. 9** Network density for the extreme Pareto points in the full instance. Left: the most passenger-friendly route set (i.e. the red route set in Fig. 7). Right: the most operator-friendly route set (i.e. the blue route set in Fig. 7). The colour coding shows the number of routes sets serving a specific connection within the network. The most operator-friendly routes set serves most connections with only one route causing many transfers. The most passenger-friendly route set has many connections served multiple times increasing the chance that passengers can travel directly to their destination

## 6 Conclusion

In this paper we introduced a new methodology for modelling and scaling down a street network to facilitate optimisation of public transport networks in a realistic way. Furthermore, we built our model using only data that is freely available from public sources. The process involves a systematic placement of nodes on junctions of a street network suitable for bus travel. The travel times between the nodes are generated from street data, the information about potential terminal nodes is derived from existing bus routes, and the travel demand was extracted from UK census data. In the initial study, we applied our techniques to the bus network of greater Nottingham.

We showed that the pre-existing public transport routes can be projected onto the generated instance network, allowing the direct comparison between optimisation results and pre-existing public routes in a reduced version of the generated instance. Additionally, we adapted an evolutionary multi-objective optimisation algorithm (NSGAI) to operate effectively on our Nottingham instance characterised by restricted terminal nodes (i.e., where buses can turn around). The comparison between our results and the actual 2011 bus routes of the study area indicate that it is possible to reduce both the average passenger travel time and the operator cost simultaneously. Our results further suggest that a bus network optimised for direct travel (i.e. for a fragmented service with many different operators) is not the most effective network for passengers or operators. Given that this work is only the first step in an incremental approach for general public transport optimisation, these results are very promising.

There are several possible directions for future work. One is to improve the instance generation, e.g. by using better resolved demand data, or by generating

asymmetric travel time matrices to better capture the impact of turn restrictions in a similar way to what was done in Perugia et al. (2011). Extending the optimisation to one or more of the four remaining phases mentioned in Sect. 1 (Vehicle Frequency Setting, Timetabling, Vehicle Scheduling and Crew Scheduling), is another. Allowing variable numbers of routes in our route sets, is also an important investigation to be carried out, as well as introducing multiple modes of public transport. In addition, many improvements could be made by changes in the objective functions. This may include techno-economic aspects such as vehicle crowding or required fleet size for more realism [see e.g. Jara-Diaz and Gschwender (2003) or Moccia et al. (2018)]. Further, changes to the objective function can also alleviate some of the less realistic constraints for valid route sets. Most notably is the constraint that all nodes have to be part of every route set, which forced us to generate a reduced instance in order to compare our optimisation results against real-world routes.

We provide the instance data as well as the real-world route set, our results and a python program for route set evaluation alongside this paper to allow other researchers to employ their own optimisation algorithm. The material is also available online under <https://data.mendeley.com/datasets/kbr5g3xmvk>.

**Acknowledgements** We are thankful for the very helpful comments and suggestions of three anonymous reviewers. This work is partly funded by the Leverhulme Programme Grant RP2013-SL-015.

**Open Access** This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

## Appendix 1: Script to generate travel time matrix

This section describes the travel time matrix generation, as mentioned in Sect. 2.4. The process is done in three steps

1. Extracting travel time information from route data.
2. Feasibility check and auto-correction.
3. Final check and output.

We had all three steps executed by a single python script. Its structure can be seen in the flow diagram in Fig. 10.

### Extracting travel time information

#### Auto-generate travel times from node positions

The basis for the first step is an ArcGIS network dataset of the available street network. We generated such a dataset from the UK Ordnance Survey integrated network layer with the procedure described in ESRI (UK) Ltd (2007). The described

procedure can also be applied to data from other sources, provided these can be converted to network datasets. Such data should be available from most national transport authorities or from local authorities. Alternatively, street data from OpenStreetMap (<https://www.openstreetmap.org>) can be used as long as it is sufficiently accurate for the study area.

A network dataset allows us to generate travel time data with ArcGIS using the Network Analyst function “Closest Facility”. This function generates routes<sup>18</sup> between the elements of one set of points (“Incidents”) and the elements of another set of points (“Facilities”).<sup>19</sup> The attribute table of the generated routes contains information about start and end points as well as the required travel time. By selecting the bus routing nodes (see Sect. 2.3.2) as both “Incidents” and “Facilities” we are able to generate routes from every node to every other node.

We only wish to use the travel times between directly connected nodes. Therefore, we need to determine which of the above generated routes are valid (connect two directly connected nodes) and which are invalid (connect two nodes which are not directly connected).

A route  $f_{ij}$  between two nodes  $n_i$  and  $n_j$  would be considered valid if it does not pass through any other node. It is therefore possible to check if  $f_{ij}$  is valid by counting the number of nodes alongside it, defined as  $A_{ij}$ . A node  $n_x$  is considered as alongside  $f_{ij}$ , if the path of  $f_{ij}$  is either directly going over  $n_x$ , or, in case  $n_x$  has been generated by snapping, over a junction represented by  $n_x$  (see Fig. 2). If only the starting node  $n_i$  and its end node  $n_j$  are alongside  $f_{ij}$  ( $A_{ij} = 2$ ),  $f_{ij}$  is valid. If there are more nodes alongside  $f_{ij}$  ( $A_{ij} \geq 3$ ),  $f_{ij}$  is not valid. Nodes are considered alongside  $f_{ij}$  if the distance between them and  $f_{ij}$  is  $s/2$  or less. This is because a node can represent junction up to  $s/2$  away from it (see Sect. 2.3.2).

The values of  $A_{ij}$  can be obtained with the ArcGIS function “Locate Features Along Routes” which gives out the nodes within a certain distance of a route.<sup>20</sup>

However, the fact there may be junctions which are represented by more than one node leads to situations where  $A_{ij} \geq 3$  although  $n_i$  and  $n_j$  are directly connected (see

<sup>18</sup> These routes are not related to the public transport routes we talked about in other sections of this paper. The reason why we use the term again here is to be consistent with the ArcGIS terminology.

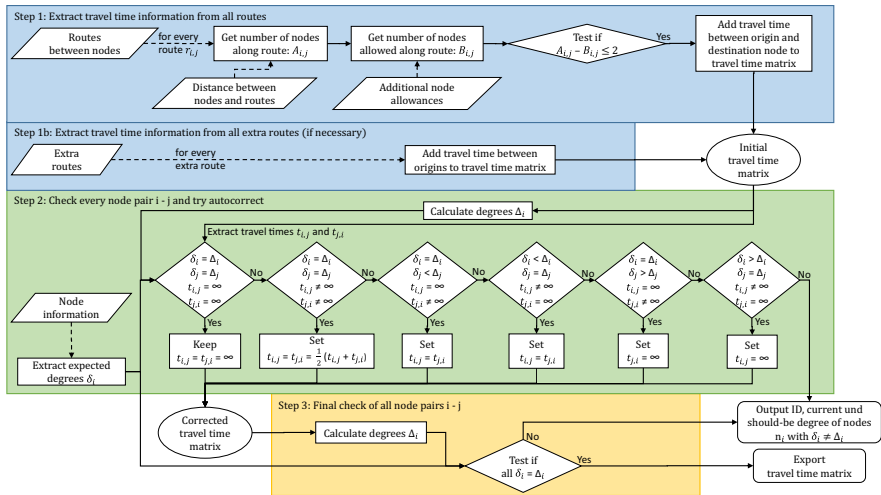
<sup>19</sup> Parameters used for “Closest Facility” function:

- Analysis settings:
  - Impedance: drive (min)
  - Facilities to find: *as many as possible*
  - U-Turns at junctions: not allowed
  - Output shape type: true shape with measures
  - Use hierarchy: yes
  - Ignore invalid locations: yes
  - Restrictions: MandatoryTurnRestrictions, OneWay, TurnRestrictions
- Accumulation attributes: drive (min)
- Network locations—finding network locations:
  - Search tolerance: snapping distance  $s$
  - Snap to: closest street-network (shape).

<sup>20</sup> Parameters used for locating features along routes function:

- Search radius: half snapping distance  $\frac{s}{2}$
- Keep only the closest route location: no





**Fig. 10** Algorithm to generate travel time matrix. The process consists of three steps from the input files generated manually and via ArcGIS. In step 1, the initial travel time matrix is generated from ArcGIS route information (see Sect. 1). Step 2 is an auto-correction procedure to fix simple mistakes in the route data based on comparisons with the expected degree of each node. This comparison is repeated in step 3 with the corrected matrix as a final check before outputting the travel-time matrix

Fig. 11). To cope with this situation, a third input file is necessary, which states how many more nodes  $B_{ij}$  are allowed alongside a route  $f_{ij}$  before  $f_{ij}$  is considered invalid. This additional input can be generated manually.

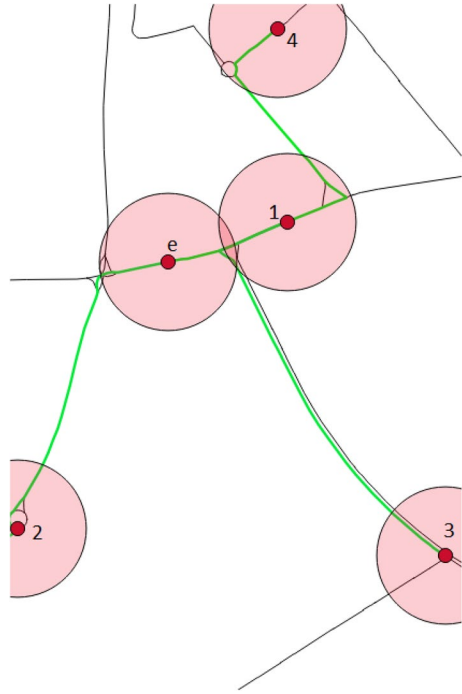
With both  $A_{ij}$  and  $B_{ij}$  known, it is finally possible to classify routes into valid and invalid. A route  $f_{ij}$  is called valid if  $(A_{ij} - B_{ij}) \leq 2$ . If this is the case, the route’s travel time is inserted in the initial OD matrix. If  $f_{ij}$  turns out to be invalid, the travel time between  $n_i$  and  $n_j$  is considered as infinite.

### Placing extra nodes

The “Closest Facilities” function generates the shortest path between two points. This, however, creates problems if the shortest path is not the most direct path (see Fig. 12). In these cases, the travel times for the direct connection has to be generated separately. In order to do so, we placed a so-called extra node along the direct connection and used the “Closest Facility” function again to generate routes from this extra node to the two nodes to be connected.<sup>21</sup> By summing up the travel time of both routes the travel time between the two nodes is generated and stored in the initial travel time matrix.

<sup>21</sup> For the generation of extra routes the same settings for the “Closest Facility” function are used as before. The only differences are that the extra nodes are used as “Incidents” and the “number of facilities to find” is set to 2.

**Fig. 11** ArcGIS generated routes (green) leading from four different starting nodes 1, 2, 3 to node e. The distance of  $\frac{\delta}{2}$  around the nodes displayed (red circle). Note that the junction between node e and node 1 is represented by both nodes. The valid routes  $f_{1,e}$  and  $f_{2,e}$  have only two nodes alongside them. The invalid route  $f_{4,e}$  has three nodes alongside it (e, 1, and 4). The route  $f_{3,e}$  has to be valid; however, it has three nodes alongside it (e, 1, and 3). In this case, an additional node allowance  $B_{3,e} = 1$  has to be used to allow  $f_{3,e}$  to be considered a valid route (colour figure online)



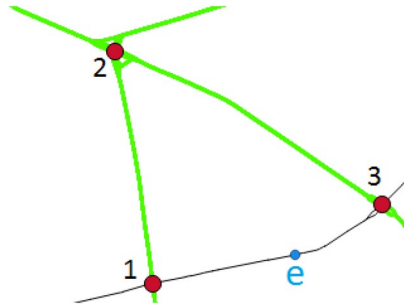
### Feasibility check and auto-correction

The “Closest Facility” function sometimes produces nonsensical routes (see Fig. 13) and it is therefore important to check if the entries in the initial travel time matrix are correct. In order to do this, we calculate the degree  $\delta_i$ , the sum of all direct neighbours, of node  $n_i$ , and compare to its expected degree  $\Delta_i$ .<sup>22</sup>

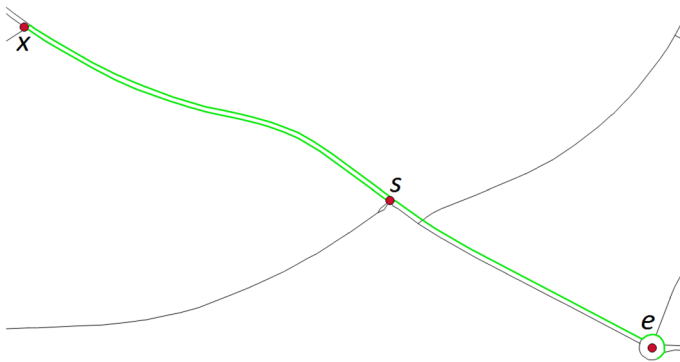
It is then possible to check for every node pair  $n_i, n_j$  if the calculated and expected degrees of both nodes match up. The result of this comparison is used for an auto-correction procedure:

1. If  $\delta_i = \Delta_i$  and  $\delta_j = \Delta_j$  and  $t_{ij} \neq \infty$  and  $t_{ji} \neq \infty$ :  
Everything correct: Both travel times are averaged to create a symmetrical matrix  $t_{ij} = t_{ji} = \frac{1}{2}(t_{ij} + t_{ji})$
2. If  $\delta_i = \Delta_i$  and  $\delta_j = \Delta_j$  and  $t_{ij} = t_{ji} = \infty$ :  
Everything correct.
3. If  $\delta_i = \Delta_i$  and  $\delta_j < \Delta_j$  and  $t_{ij} \neq \infty$  and  $t_{ji} = \infty$ :  
 $f_{ji}$  seem to have been wrongly classified as invalid: set  $t_{ji} = t_{ij}$
4. If  $\delta_i < \Delta_i$  and  $\delta_j = \Delta_j$  and  $t_{ij} = \infty$  and  $t_{ji} \neq \infty$ :  
 $f_{ij}$  seem to have been wrongly classified as invalid: set  $t_{ij} = t_{ji}$
5. If  $\delta_i = \Delta_i$  and  $\delta_j > \Delta_j$  and  $t_{ij} = \infty$  and  $t_{ji} \neq \infty$ :

<sup>22</sup> The expected degree  $\Delta_i$  of a node  $n_i$  has to be generated manually.



**Fig. 12** ArcGIS generated routes (green) between several nodes. The shortest path both between the nodes 1 and 3 goes over node 2. The street connecting node 1 and node 3 is not used by the “Closest Facility” function. In order to extract the travel times along this connection an extra node  $e$  is inserted. This allows to generate the routes  $f_{e,1}$  and  $f_{e,3}$  and to sum up their travel time to  $t_{1,3} = t_{e,1} + t_{e,3}$  (colour figure online)



**Fig. 13** ArcGIS generated route (green) from node  $s$  to node  $e$  leading over node  $x$  for no identifiable reason. (Node  $s$  is clipped to a to center of a junction allowing turns in both directions.) (colour figure online)

- $f_{ji}$  seem to have been wrongly classified as valid: set  $t_{ji} = \infty$
- 6. If  $\delta_i > \Delta_i$  and  $\delta_j = \Delta_j$  and  $t_{ij} \neq \infty$  and  $t_{ji} = \infty$ :  
 $f_{ij}$  seem to have been wrongly classified as valid: set  $t_{ij} = \infty$
- 7. If none of the above:

An issue that has not been caused by ArcGIS but by mistakes in one of the input files. The script outputs the ID, calculated degree and expected degree of the nodes where  $\delta_i \neq \Delta_i$  to help identify the source of the problem.

### Final check and output

Finally the degrees of all nodes are calculated and compared against the expected degrees. If  $\delta_i = \Delta_i$  for all nodes  $i$ , the final travel time OD matrix is generated. Otherwise, the script outputs the list of nodes with  $\delta_i \neq \Delta_i$  which serves as a basis for adjustments on the inputs for the next run of the script. Further, a lines shape file for

all valid connections is generated to allow a separate visual check if all connections were generated correctly. (These shape files were used in the generation of Fig. 6 and the right side of Fig. 1.)

## Appendix 2: Mutation operations

The following sections outline the mutation operations “Delete nodes” and “Add nodes” as well as the “Add missing nodes” repair operation, all used as part of the GA. The other mutation operations “Exchange”, “Replace”, and “Merge” as well as the repair operation “Replace Overlapping” are not described here as our implementation of these operations does not differ significantly to earlier descriptions of these algorithms e.g. in John (2016) and Mumford (2013).

As the operators described here are quite complex, we advise following the provided flow diagrams while reading the explanatory text.

### *Terminology*

Before starting the description of the algorithms we need to introduce some terms we use during the description:

- “List”: Described is a randomised list of all possible elements (e.g. a route list is a list of all routes in a route set in a randomised order.)
- “Select from list”: takes the first entry from a list and thereby returns the entries in a pseudo-random order. The selected entries are removed from the list so the list is empty once all elements have been selected.
- “Reset list”: reinserts all previously selected elements back into the list and reshuffles it.
- “Reverse route”: Routes are lists of nodes which can be changed to reverse order (e.g.  $[n_1, n_2, n_3]$  to  $[n_3, n_2, n_1]$ ). As it is assumed that routes are travelled in both directions reversing a route does not change the connectivity in the route set.
- “Reaching a terminal in  $X$  steps”: “Add nodes”- and “Add missing nodes”-operation require information about how many steps a node  $n_i$  is away from the next terminal node. (Each step means passing another node.) This information can be calculated from the adjacency matrix in advance.
- The maximal number of steps possible  $X_{max}$ : Determines the maximal number of steps allowed in “Add nodes”- and “Add missing nodes”-operation.  $X_{max}$  is set as the largest number of steps between any node in and the nearest terminal node in the instance network.

### Delete nodes

The “Delete nodes” operator was first described in Mumford (2013) and used to delete nodes from the end of randomly selected routes. However, this process needs to be more complex if a route has to end on a terminal node. Figure 14 shows the flow diagram of the “Delete nodes” mutation operation adapted for the use with terminal nodes.

At the beginning of the operation  $Z \in [0, \frac{l_{max}}{2}]$  is determined at random.  $Z$  is the minimal number of nodes to be deleted in the entire route set.

After  $Z$  is set, the routes in the selected route set are sorted into a random order, and the first route of this list is selected as  $r$ . It is first checked if  $r$  includes more than two terminal nodes, as otherwise deleting one node would make  $r$  invalid. If  $r$  includes enough terminals, a copy  $r_{orig}$  is made and one node after another is deleted until  $r$  again ends on a terminal node. If  $len(r) \geq l_{min}$  is still true, the shortened route is accepted and reinserted into the route set. If  $r$  becomes too short, the original route  $r_{orig}$  is restored. If the route has not yet been reversed, it is reversed now and a new attempt to delete nodes is started with the reversed route. If deleting nodes in the reversed route again leads to a too short route, the next route in the route list is selected.

This process is repeated until at least  $Z$  nodes have been deleted from the total route set or all routes have been tried out.

### Add nodes

The “Add nodes” operator was first described in Mumford (2013) and added adjacent nodes at the ends of randomly selected routes. However, this process needs to be more complex if a route has to end on a terminal node. This new version uses a guided random walk to connect a given route to the next possible terminal node. This process ensures that the “Add nodes” operation is balanced with the “Delete nodes” operation (which deletes nodes at least until it reaches the next terminal). Figure 15 shows the flow diagram of the “Add nodes” mutation operation.

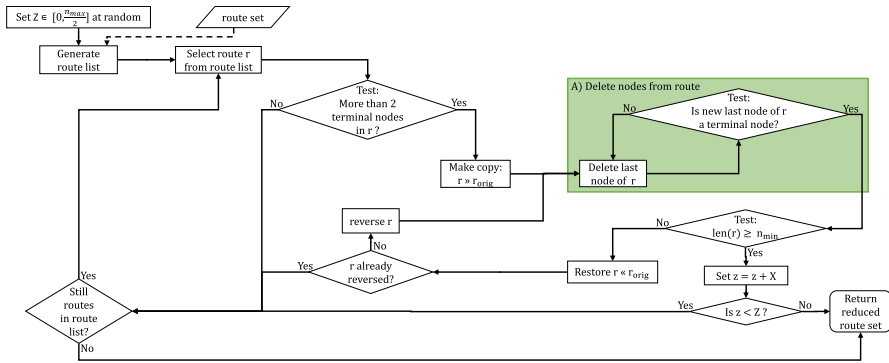
At the beginning of the operation  $Z \in [0, \frac{l_{max}}{2}]$  is determined at random.  $Z$  is the minimal number of nodes to be added in the entire route set.

After  $Z$  is set, the routes in the route set are arranged in random order, and the first route of this list is selected as  $r$ . A copy  $r_{orig}$  made from it, and a step counter  $X$  set to  $X = 1$ . It is then tested if  $X$  nodes can be added to  $r$  (if  $len(r) + X \leq l_{max}$ ). If this is true, the algorithm checks if there are terminal nodes  $V_t$  (at least one), which are not yet part of  $r$  and which can be reached in  $X$  steps from  $r$ 's current last node  $n_l$ . If no  $V_t$  can be found, the step counter is increased to  $X = X + 1$  and, it is again checked if terminal nodes  $V_t$ , fulfilling the above mentioned conditions, do exist. This process is repeated as long as  $X \leq X_{max}$  and  $len(r) + X \leq l_{max}$  is true.

If nodes  $V_t$  do exist, but there is more than one step needed to reach them ( $X > 1$ ), it is tried to close the gap by appending other nodes to  $r$ . As the success of this process is not guaranteed, a copy  $X_{orig}$  is made from  $X$  to be able to restore it later.

The nodes to close the gap are selected via a guided random walk: It is tested if there are nodes  $N_k$  adjacent to  $r$ 's last node  $n_l$ , which are not yet part of  $r$  and can reach a node in  $V_t$  in  $X - 1$  steps. If node(s)  $N_k$  exist, one of them is selected at random and is appended to  $r$ , thereby becoming the new last node  $n_l$  and  $X$  is reduced by one. This process is repeated until  $X = 1$  is reached. When this happens, one of the terminal nodes in  $V_t$  which is adjacent to  $r$ 's current last node  $n_l$  is selected at random and appended to  $r$ . Route  $r$  is now again valid (it ends again on a terminal node).

If node(s)  $N_k$  do not exist (because all potential nodes are already used within  $r$ ), the original route  $r_{orig}$  is restored, the step counter is set to  $X = X_{orig} + 1$ .



**Fig. 14** Flow diagram of “Delete nodes” mutation operation: the operation begins with the selection of a route from a randomised route list. If the routes include more than two terminal nodes, the operation starts to delete nodes until either the route becomes too short or it ends again on a terminal node (box A). If the resulting route is too short, the original route is restored and reversed and the process started again. If the attempt fails, a new route is selected

If either the step counter reaches the maximal level ( $X > X_{max}$ ) or  $r$  would get too long if  $X$  nodes would be added to it, the attempts to append nodes to this route are stopped. If the route has not yet been reversed, it is now reversed and the whole process starts again with a new last node  $n_l$ . However, if the reversed route can not be extended, a new route is selected.

The entire process is repeated until either at least  $Z$  nodes have been added to the route set or all routes have been tried out.

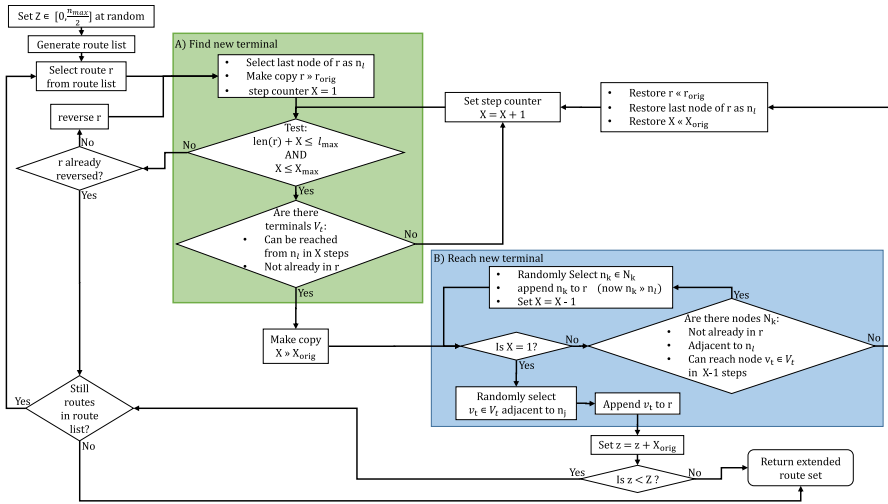
### Add missing nodes

A repair operation which reinserts missing nodes back into the route set was first introduced in Mumford (2013). However, the original version only added the missing nodes to randomly selected routes. This process is not sufficient if a route has to end on a terminal node.

To take the constraint of terminal nodes into account, this version has two phases: The first phase ensures that all terminal nodes are included in at least one route. It also tries to reconnect as many other missing nodes as possible in the process. The flow diagram of the first phase is shown in Fig. 16. The second phase tries to insert remaining missing non-terminal nodes in randomly selected routes. Its flow diagram is shown in Fig. 17.

### Add missing terminal nodes

The first phase starts with a step counter  $X$  set to  $X = 1$ , the generation of a randomised route list and the selection of a route  $r$  from that list. It is tested if  $r$  can be extended by  $X$  nodes. If yes, it is further tested if there are missing terminal nodes  $V_l$



**Fig. 15** Flow diagram of the “Add nodes” mutation operation: the operation begins with the selection of a route  $r$  from a randomised route list and then testing if  $X$  nodes can be added to  $r$  and if there are possible new terminal nodes  $V_t$  within  $X$  steps (box A). If node(s)  $V_t$  are found but cannot be reached directly, the additional steps are filled with other nodes  $n_k$ , each one step closer to the new terminal (box B). If no suitable nodes can be found, the original route is restored and the step counter is increased. If too many steps would be required,  $r$  is first reversed and the process started again. If this attempt also fails, a new route is selected

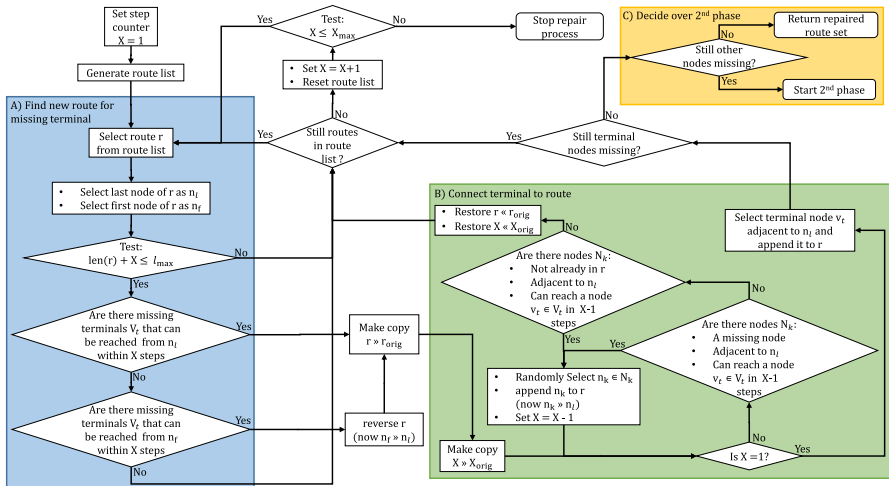
which can be reached in  $X$  steps from either  $r$ 's last node  $n_l$  or  $r$ 's first node  $n_f$  of  $r$ . If no  $V_t$  exists, or if  $r$  would become too long, a new route is selected.

If  $V_t$  exists but there is more than one step needed to reach a node in  $V_t$ , other nodes are appended to  $r$  to close the gap. At this point a copy  $X_{orig}$  is made from  $X$  to be able to restore it later. (If  $V_t$  can only be reached from  $n_f$   $r$  is reversed so that  $n_f$  becomes  $n_l$ .)

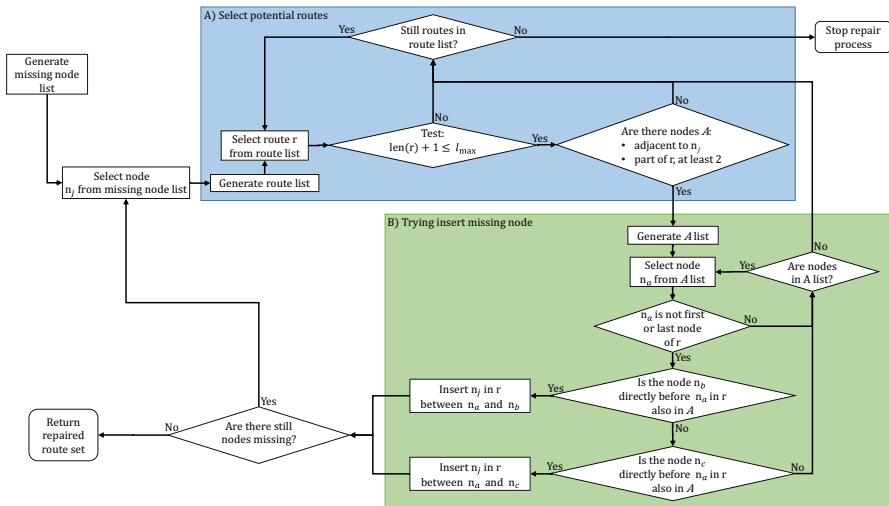
It is then tested if there are nodes  $N_k$  which are adjacent to  $r$ 's current last node  $n_l$  and can reach a node in  $V_t$  in  $X - 1$  steps. Furthermore, these adjacent nodes cannot be already part of  $r$ . If suitable adjacent nodes exist, one such node  $n_k$  is selected at random and appended to  $r$ , thereby becoming the now  $n_l$ , and  $X$  is reduced by one. Missing nodes are preferred to speed up the repair process. This process is repeated until  $X = 1$ . Now  $V_t$  can be reached directly and one node  $v_t \in V_t$  which is adjacent to  $n_l$  is selected at random and appended to  $r$ .

After a missing terminal node  $v_t$  has been successfully connected to a route, it is checked to see if further terminal nodes are missing. If yes, the next route from the route list is selected and the process starts again. If all routes have been tried, the step counter  $X$  is increased by one and the route list is reset. The process then starts again, checking for missing terminals one step further away from the end nodes of  $r$ .

If  $X$  reaches  $X_{max}$  before all terminal nodes could be connected, the process stops and the route set is returned as not repairable. If all missing terminal nodes can be connected to routes, it is tested if there are other (non-terminal) nodes



**Fig. 16** Flow diagram of the first phase of “Add missing nodes”: the process starts a step counter set to  $X = 1$  and the selection of a route  $r$  from a route list. It is tested if there are missing terminal nodes  $V_i$  within  $X$  steps from either the first or last node of  $r$  (box A). If at least one such node is found, it is either inserted directly, or, in case of  $X > 1$ , the additional steps are filled with other nodes, each one step closer to the missing terminal node. Other missing nodes are prioritised in this process (box B). If at least one missing terminal node cannot be connected to any route within  $X$  steps,  $X$  is increased by one and the routes are tried again. Once there are no missing terminal nodes the second phase starts, or, if there are no other missing nodes left, the repair process ends (box C)



**Fig. 17** Flow diagram of the second phase of “Repair nodes”: the process starts with selecting a missing node  $n_j$  and a route  $r$  from randomised lists. It is tested if the route could take another node and if there is an overlap of at least two nodes between the nodes currently in  $r$  and the once adjacent to  $n_j$  (box A). If yes, it is tested if two of these nodes are consecutive nodes in  $r$  so  $n_j$  can be inserted in between them (box B). If such a combination cannot be found, a new route is selected until either all missing nodes are inserted in routes or there are no routes that were not tried



missing. If yes, the second phase of the repair procedure is started. If not, the repaired route set is returned to the GA.

### Add remaining missing nodes

The second phase starts with the generation of a randomised route list and the selection of one node  $n_j$  at random from the missing nodes. Next, a route  $r$  is selected from the route list and is tested if it got too long if one node was inserted. If inserting a node is possible, it is tested if there is a group  $A$  of at least two nodes which is part of  $r$  and also adjacent to  $n_j$ . If no overlap can be found or  $r$  would get too long, a new route is selected.

If  $A$  can be found, one node  $n_a \in A$  is selected at random. If  $n_a$  is neither the first nor the last node of  $r$ , it is tested if either the node  $n_b$  which is in  $r$  directly before  $n_a$ , or the node  $n_c$  which is in  $r$  directly after  $n_a$  is also in  $A$ . If one of these is the case, the node  $n_j$  is inserted into  $r$  either in between  $n_a$  and  $n_b$  or between  $n_a$  and  $n_c$ . If there are further nodes missing, a new node  $n_j$  is selected and the route list is reset. Otherwise the route set is returned as repaired.

If there is one node  $n_j$  that cannot be inserted into any route, the route set is returned as not repairable.

## References

- Ahmed L, Mumford CL, Kheiri A (2019) Solving urban transit route design problem using selection hyper-heuristics. *Eur J Oper Res* 274(2):545–559
- Amiripour SM, Ceder AA, Mohaymany AS (2014) Designing large-scale bus network with seasonal variations of demand. *Transp Res Part C Emerg Technol* 48:322–338
- Ammons DN (2001) *Municipal benchmarks: assessing local performance and establishing community standards*, 2nd edn. SAGE Publications, Thousand Oaks
- Arbex RO, da Cunha CB (2015) Efficient transit network design and frequencies setting multi-objective optimization by alternating objective genetic algorithm. *Transp Res Part B Methodol* 81:355–376
- Baaj MH, Mahmassani HS (1991) An ai-based approach for tansit route system planning and design. *J Adv Transp* 25:187–209
- Bagloee SA, Ceder AA (2011) Transit-network design methodology for actual-size road networks. *Transp Res Part B Methodol* 45(10):1787–1804
- Bielli M, Caramia M, Carotenuto P (2002) Genetic algorithms in bus network optimization. *Transp Res Part C Emerg Technol* 10(1):19–34
- Ceder A, Wilson NHM (1986) Bus network design. *Transp Res B* 20B(4):331–344
- Cipriani E, Gori S, Petrelli M (2012) Transit network design: a procedure and an application to a large urban area. *Transp Res Part C Emerg Technol* 20(1):3–14
- Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans Evol Comput* 6(2):182–197
- ESRI (UK) Ltd (2007) Using OS MasterMap Integrated Transport Network (ITN)<sup>TM</sup> Layer with ArcGIS . ESRI (UK) White Paper
- Fan L, Mumford CL (2010) A metaheuristic approach to the urban transit routing problem. *J Heuristics* 16(3):353–372
- Feng C, Hsieh C, Peng S (2010) Optimization of urban bus routes based on principles of sustainable transportation. *J East Asia Soc Transp Stud* 7:1137–1149
- Golding J (2018) Best practices and methodology for OD-matrix creation from CDR-data. Technical report, University of Nottingham, Business School, N-LAB

- Guierrez-Jarpa G, Laporte G, Marianov V, Moccia L (2017) Multi-objective rapid transit network design with modal competition: the case of Concepción, Chile. *Comput Oper Res* 78:27–43
- Jara-Diaz SR, Gschwender A (2003) Towards a general microeconomic model for the operation of public transport. *Transp Res* 23(4):453–469
- John MP (2016) Metaheuristics for designing efficient routes & schedules for urban transportation networks. Ph.D. thesis, University of Cardiff
- John MP, Mumford CL, Lewis R (2014) An improved multi-objective algorithm for the urban transit routing problem. In: Blum C, Ochoa G (eds) *Evolutionary computation in combinatorial optimisation*. Springer, Berlin, pp 49–60
- Kiliç F, Gök M (2014) A demand based route generation algorithm for public transit network design. *Comput Oper Res* 51:21–29
- Leblanc LJ (1975) An algorithm for the discrete network design problem. *Transp Sci* 9(3):183–199
- Mandl CE (1979) *Applied network optimization*. Academic Press, Cambridge
- Mauttone A, Urquhart ME (2009) A route set construction algorithm for the transit network design problem. *Comput Oper Res* 36(8):2440–2449
- Moccia L, Allen DW, Bruun EC (2018) A technology selection and design model of a semi-rapid transit line. *Public Transp* 10(3):455–497
- Mumford CL (2013) New heuristic and evolutionary operators for the multi-objective urban transit routing problem. In: 2013 IEEE congress on evolutionary computation. IEEE, pp 939–946. <https://ieeexplore.ieee.org/abstract/document/6557668>
- Nayem MA, Rahman MK, Rahman MS (2014) Transit network design by genetic algorithm with elitism. *Transp Res Part C Emerg Technol* 46:30–45
- Nielsen G, Nelson J, Mulley C, Tegner G, Lind G, Lange T (2005) HiTrans Best Practice Guide 2: Public transport – planning the networks. HiTrans. <https://www.crow.nl/downloads/documents/13359>
- Pattnaik S, Mohan S, Tom V (1998) Urban bus route network design using genetic algorithm. *J Transport Eng* 124(4):368–375
- Perugia A, Moccia L, Cordeau JF, Laporte G (2011) Designing a home-to-work bus service in a metropolitan area. *Transp Res Part B Methodol* 45(10):1710–1726
- Poorzahedy H, Safari F (2011) An ant system application to the bus network design problem: an algorithm and a case study. *Public Transp* 3(2):165–187
- Shih M, Mahmassani H (1994) A design methodology for bus transit networks with coordinated operations. Technical report, University of Texas, Center for Transportation Research
- Silman LA, Barzily Z, Passy U (1974) Planning the route system for urban busses. *Comput Oper Res* 1(2):201–211
- Soehodo S, Koshi M (1999) Design of public transit network in urban area with elastic demand. *J Adv Transp* 33(3):335–369
- Turfitt R (2018) Statutory Document No. 14 Local bus services in England (outside London) and Wales. Technical report, Senior Traffic Commissioner (UK)
- UK Department for Transport (2015) National public transport data repository. <https://data.gov.uk/dataset/nptdr>. Accessed 6 June 2019
- UK Office for National Statistics (2016a) Census geography. <http://www.ons.gov.uk/ons/guide-method/geography/beginner-s-guide/census/index.html>. Accessed 6 June 2019
- UK Office for National Statistics (2016b) Census output areas population weighted centroids 2011. [https://geoportal.statistics.gov.uk/datasets/ba64f679c85f4563bfff7fad79ae57b1\\_0](https://geoportal.statistics.gov.uk/datasets/ba64f679c85f4563bfff7fad79ae57b1_0). Accessed 6 June 2019
- Wills M (1986) Gravity-opportunities trip distribution model. *Transp Res Part B Methodol* 20(2):89–111
- Wilson AG (1969) The use of entropy maximising models, in the theory of trip distribution, mode split and route split. *J Transp Econ Policy* 3(1):108–126
- Zhang Y, Qin X, Dong S, Ran B (2010) Daily O-D matrix estimation using cellular probe data. In: TRB 89th annual meeting compendium of papers DVD, Transport Research Board. [http://odd.topslab.wisc.edu/publications/2010/Daily%20O-D%20Matrix%20Estimation%20using%20Cellular%20Probe%20Data%20\(10-2472\).pdf](http://odd.topslab.wisc.edu/publications/2010/Daily%20O-D%20Matrix%20Estimation%20using%20Cellular%20Probe%20Data%20(10-2472).pdf)
- Zhao F, Gan A (2003) Optimization of transit network to minimize transfers. Technical report, Florida International University, Department of Civil and Environmental Engineering, Transport Research Board. <https://trid.trb.org/view/697978>