Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 151 (2019) 834–839

Procedia Computer Science

www.elsevier.com/locate/procedia

The 8th International Workshop on Agent-based Mobility, Traffic and Transportation Models, April 29 - May 2, 2019, Leuven, Belgium

Improving speed and realism of an evolutionary minibus network design process

Gregor Leich[a,*], Kai Nagel[a]

[a]*Transport System Planning and Transport Telematics, Technische Universität Berlin, Salzufer 17-19, Berlin 10587, Germany*

Abstract

This paper deals with improving an existing evolutionary minibus network design process. Computation time was reduced by using a more suitable public transit routing algorithm. Furthermore, a minibus stop creator was developed which attempts at identifying road junctions and putting stops only on approach links and between junctions. Lastly, the circuity of the minibus routes returned was reduced by applying penalty scores for certain route circuity and complexity measures.

Keywords: transit network design; minibus; evolutionary algorithm; MATSim; public transit routing; circuity; public transit

1. Introduction

Informal minibus public transit systems operate in many cities of the world. Neumann [3] aims at modelling such informal public transit systems and creating an evolutionary process for public transit network design. The following paper introduces that design process and deals with some proposed improvements in terms of computational performance and realism of the output minibus transit network. The major bottleneck for computation time was the public transit routing algorithm in combination with a high density of stops which forces to check many possible transfer locations. Later the paper deals with reducing the seemingly unrealistic level of circuity and complexity of the minibus lines created by the design process.

2. Evolutionary minibus network design process

The work presented in this paper is based upon the minibus network design process developed by Neumann [3] which is an extension to the multi-agent transport simulation MATSim[2]. A typical MATSim model contains at least

* Corresponding author. Tel.: +49-30-314-28666 ; fax: +49-30-314-26269.
E-mail address: leich@vsp.tu-berlin.de

10.1016/j.procs.2019.04.114

a transport network and a synthetic population with daily plans. In an iterative cycle the three principal steps *modification of daily plans* (“replanning”), *execution of daily plans* in a traffic flow simulation (“mobsim”) and *scoring of the executed plans* (“scoring”) are repeated for typically several hundred or thousand iterations. During the replanning step some agents change e.g. the mode of transport, the departure time and the route they take on the transport network. In the mobsim step the selected daily plans are executed so travel times and similar information can be calculated. Afterwards the performance of the executed plans is evaluated in the scoring step. By this cycle, the agents of the synthetic population can optimize their daily plans while moving towards a stochastic user optimum. A formal public transit system can be included in the simulation by providing a transit schedule.

The minibus extension[3] (called minibus contrib) extends MATSim by adding minibus operators which have plans that contain minibus transit services. The extension interacts with MATSim by modifying the transit schedule before each iteration. During the replanning step some agents of the synthetic population might replan to use one of the minibus services contained in the updated transit schedule. In the scoring step minibus operators score their executed plan (and the individual minibus routes included in it) based upon operating cost and revenue generated from passengers. After a certain number of iterations minibus operators incurring losses are removed. New minibus operators are added as long as a certain share of profitable operators among all operators is fulfilled.

Before each iteration begins, minibus operators can modify their plans e.g. by redistributing vehicles between minibus routes, removing unprofitable minibus routes and adding new routes. New minibus routes are based upon a well performing existing route of the same operator. They can have a different time of operation, stops which generated little revenue can be removed or new stops can be added at the ends of the route or between two existing stops of the route. Every new operator starts with a new minibus route between two stops selected by weighted randomness from a list of minibus stops weighted by the number of neighbouring activities of the synthetic population. A plan for a minibus route contains a list of stops that should be served, but this is only a subset of all stops served. The itinerary on the road network between these stops to be served is selected using a shortest travel time path search and all possible minibus stops found on that path are served, too. Possible minibus stops are put at the ends of all road network links accessible to minibusses which pass certain filter criteria such as minimum link capacity or maximum link freespeed.

3. Speeding up passenger routing

All agents of the synthetic population who intend to use public transit have to be routed over the public transit network. Since the public transit network changes as the minibus network evolves, agents have to be rerouted several times over the course of the iterations. Unfortunately public transit routing in MATSim tended to take up a very large share of the total computation time. Furthermore, the larger the public transit network the larger is the computation time for routing. This made transit routing the major bottleneck that could render the minibus network design process described in section 2 impractical for large MATSim scenarios with several hundred thousand agents due to the unacceptably high computation time.

Issues with the default public transit router. The default transit router in MATSim is based on a Dijkstra approach.[4] A transit network graph is created in which every TransitRoute (a data structure for a public transit route in MATSim) is represented by a series of links connecting all stops served by that line. In addition to that, (directed) transfer links are created for each pair of transit stops within a certain maximum transfer distance and for each pair of transit routes serving these stops. As an example, assume stops A, B and C are within maximum transfer distance from each other, route 1 serves stop A and stop B and route 2 serves stop A and stop C. In this case transfer links are created from route 1 to route 2 from stop A to A, from A to C, from B to A and from B to C. Thereby, the transit router has to consider the essentially identical transfer from route 1 to route 2 four times. Since junction areas in the road network can consist of many short links, similar situations occur frequently. Even worse, as the minibus network expands, more and more minibus routes will serve stops in the junction area and will cause a tremendous increase in the number of transfer links. In the above mentioned example a new route 1a serving stops A and B would cause the number of transfer links to increase from $4 \times 2 = 8$ to $4 \times 2 \times 3 = 24$ (4 transfer links for each pair of two routes).

For one larger scenario of the city of Cape Town with about 300 000 agents this lead to a significant increase in computation time by roughly 1 min from iteration to iteration as the minibus network expanded. The replanning step, which runs the transit router, took about 340 s out of a total computation time of 428 s in the fifth iteration.

In comparison to the first iteration, this is 300 s more for replanning and only 24 s more for other iteration steps. Furthermore, the excessive number of transfer links forces to assign much memory to the router.

Approaches to reduce computation time. Two improvements were implemented to reduce routing times: First, the creation of minibus stops was altered to reduce the number of minibus stops in junction areas. Second, the transit router was replaced with a RAPTOR implementation for MATSim. The RAPTOR transit router is not Dijkstra-based, instead, it operates in rounds with one round per transfer.[1] Most importantly it traverses every route at most once per round.[1] Thereby, the number of possible transfer locations as shown in the example above is of less importance for computational performance. Fortunately, a RAPTOR implementation for MATSim became available in 2018,[5] so work concentrated on the interaction between the minibus extension and the external MatsimSBBExtensions[6] project which includes the RAPTOR implementation. The solution implemented here was that the minibus extension throws a `TransitScheduleChangedEvent` that is caught by the (updated) MatsimSBBExtensions and triggers the RAPTOR router to update the transit schedule data. Using the RAPTOR router (still on the old minibus stops) reduced replanning time in the fifth iteration from about 340 s to less than a tenth and total computation time to about a fifth of using the default router. Furthermore, the increase in replanning time from iteration to iteration is far lower at about 5 s (per iteration).

New minibus stop creator. For the creation of possible minibus stops a new class `CreatePStopsOnJunctionApproachesAndBetweenJunctions` based on the existing class `CreatePStops` was developed. It differs from the existing class in using a simplified road network, which is used to determine the position of junction areas. For the network simplification it employs the `IntersectionSimplifier` package by Johan W. Joubert from the University of Pretoria. His algorithm uses a density-based clustering approach as published by Zhou et al. [7]. It builds clusters of road network nodes which have a certain minimum number of nodes and a certain maximum distance to another node of the cluster. Junction areas tend to consist of many nodes with short distances between them whereas on the road sections between junctions the density of nodes tends to be lower and distances between them tend to be higher. For each cluster all nodes are merged, all links between merged nodes are removed and the links entering or exiting the cluster are reconnected to a new central cluster node.

Applying this intersection simplifier can reduce the number of links and nodes in junction areas drastically. The bus stop creator then removes duplicate links, i.e. links between two nodes which are already directly connected by another link are removed. Finally the `NetworkSimplifier` class is run and removes nodes with only one incoming and one outgoing link and merges those links. In consequence, all remaining nodes are located at an intersection where bus routes can have different paths, i.e. bus routes passing that node can arrive from different links or can leave the intersection on different links. In order to ensure that transferring between bus routes is possible, it seems reasonable to place stops on all arriving links.

As a first approach minibuses were run on the simplified road network with stops on all remaining links. However, the simplified network can be very distorted, so the travel times calculated in the mobsim step (see section 2) can be distorted, too. Therefore the stop creation algorithm was altered to use the simplified network only to determine on which links of the original network minibus stops should be put. The objective was to avoid links inside junction areas and put stops on junction area approaches and between junction areas instead.

Each remaining link in the simplified network is assumed to connect two junction areas. Using the link ID the original link on the original network is searched. The algorithm then walks starting from that link on the original network in the direction of travel over all nodes which have exactly one incoming and one outgoing link until it finds the first real intersection node with multiple incoming or multiple outgoing links. It assumes that this is the beginning of the junction area and puts a stop on the link approaching that node. Then it walks backwards and puts in-fill stops at a certain interval until it reaches the next intersection node, assuming that this is the previous junction area. It does not put a stop on that last link to serve the previous junction area, rather all stops are located at the end of the corresponding link and links leaving junction areas only have stops if the in-fill stop interval implies a stop at the opposite end of the link. As a result, the number of minibus stops created dropped from 11 120 to 6 670 in the Cape Town scenario. More importantly, the number of stops within junction areas and thereby the number of transfers between bus routes to be considered is reduced (see figure 1 where only red links have stops). This further accelerated the routing by a factor of about two when using the RAPTOR router.

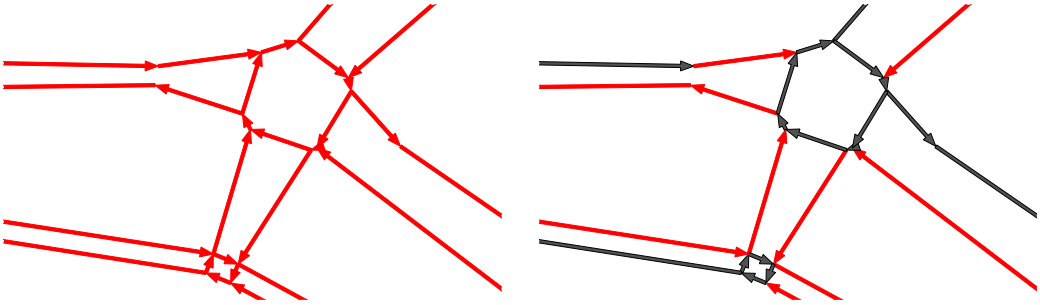

Fig. 1. Original minibus stops (left) and new minibus stops (right). Arrows indicate the direction of travel of the links. Stops are located on the red coloured links at the position of the arrow.

Overall. The reduced number of stops in combination with the RAPTOR router allowed to run the above mentioned Cape Town scenario for 1 500 iterations within a few days. In the 5th iteration the replanning step (which re-routes some public transit and minibus passengers) was about 25 times faster and the overall simulation about 7 times faster. More importantly, the increase in computation time from iteration to iteration diminished from about 1 min to less than 10 s, so the speed gains are probably higher considering a run of several hundred iterations. Furthermore, even in the last iterations when the minibus network has reached its highest number of routes, replanning took up less than half of total computation time per iteration.

4. Penalties for unrealistic, circuitous minibus lines

The minibus routes created by the evolutionary minibus network design process descibed in section 2 tend to look very complicated. Whereas bus routes in reality often run on a rather straight path and typically do not pass over the same road more than once per direction, the algorithm tends to create many complex bus routes which are very circuitous and run over the some links more than once (see figure 2 for an example). Furthermore, instead of running back and forth on the same road in both directions the evolutionary algorithm tends to add multiple side trips and choose different parallel roads per direction. Many lines form rather an oval shape than a straight line. Even though there might be some cases in which bus routes with such characteristics might exist in reality, it seems unrealistic that the majority of all bus routes shows these characteristics.

An option would be to prevent such circuitous bus routes from being created in the first place, e.g. by improving the replanning strategies. However, a simpler and more straightforward approach in line with the evolutionary method is to include penalties for circuitous routes into the scoring of minibus plans. The score calculated for each minibus route is the sum of all revenue minus the sum of all operating costs of that minibus route. Penalties are now implemented as an additional (monetary) cost that reduces the score respectively the profit of the route. Penalties will trigger the operator to move vehicles to other more profitable routes and sell vehicles to make up for the monetary loss incurred. If no vehicle is left over, the route is removed.

Four different penalty functions were implemented:

1. `Stop2StopVsTerminiBeelinePenalty`: Sum of all beeline distances of consecutive stops divided by the beeline distance between the two terminus stops. This function reduces side trips and obtain straighter more direct routes.
2. `AreaBtwLinksVsTerminiBeelinePenalty`: Area between all links divided by the beeline distance between the two terminus stops. This function reduces loop shaped route segments and obtains routes going back and forth in the same narrow corridor.
3. `StopServedMultipleTimesPenalty`: Number of stops on the route divided by the number of unique (physical) stop facilities served. This function is supposed to reduce overlapping line segments, i.e. the route passes multiple times over the same links.

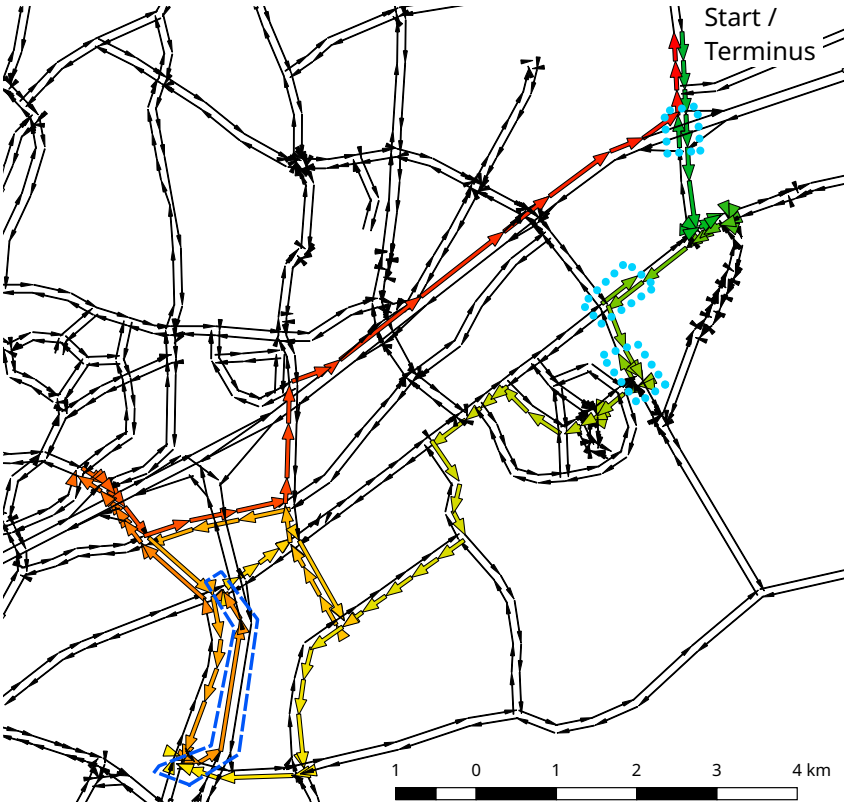

Fig. 2. Example for a circuitous minibus route produced by the evolutionary minibus network design process without penalties for circuity. Arrows denote the direction of the links, not the stops served. The minibus route starts on the dark green links. The colour of the traversed links changes via yellow to red on the terminus link. In the areas marked in blue the line overlaps itself, i.e. it runs multiple times on the same links.

A maximum acceptable penalty value (valueToStartScoring), different for each penalty type, is substracted from the preliminary penalty obtained from the division described above. If the result is positive, the penalty exceeds the maximum acceptable penalty and the result of the substraction is multiplied with a cost factor that weights the resulting penalty score against the fare and cost rates set for minibuses. Otherwise no (in this case negative) penalty is applied. There is a separate cost factor for each penalty type. The corresponding pseudo code for the `Stop2StopVsTerminiBeelinePenalty` is shown below:

```
penalty = Stop2StopVsTerminiBeelinePenalty - Stop2StopVsTerminiBeelineValueToStartScoring
if (penalty > 0) {score = Stop2StopVsTerminiBeelineCostFactor * penalty}
else {score = 0}
```

Table 1. Parameters for penalty functions applied in the Cape Town scenario and mean penalty scores per minibus route for a run without applying penalties in minibus route scoring and another run with applying penalties.

penalty function	valueToStartScoring	costFactor	mean penalty per resulting minibus route of a ... ... run without penalties	... run with penalties
`Stop2StopVsTerminiBeelinePenalty`	2.6	-800 ZAR	-8644.44 ZAR	-83.12 ZAR
`AreaBtwLinksVsTerminiBeelinePenalty`	60 m	-800 ZAR/m	-2174192.72 ZAR	-0.12 ZAR
`StopServedMultipleTimesPenalty`	1.04	-20 000 ZAR	-3548.74 ZAR	-15.76 ZAR

In a later step the score calculated above is weighted with the number of vehicles in operation, because otherwise long minibus routes with high passenger volume (but not necessarily high vehicle occupancy) could compensate the

penalty more easily. In order to restrict the influence of a particularly circuitous route on other routes of the same operator, the penalty is capped at the amount of money the operator can recover by closing the route and selling all vehicles operating on it. Thereby, the operator does not have to sell vehicles of other routes to make up for the losses incurred.

Many parameter combinations for valueToStartScoring and costFactor were tested until settling for the values in table 1 which appeared to give reasonable results. `AreaBtwLinksVsTerminiBeelinePenalty` and `Stop2StopVsTerminiBeelinePenalty` turned out to be particularly effective in reducing the circuity of the minibus routes whereas the effect of `StopServedMultipleTimesPenalty` is lesser.

Overall, the number of minibus routes as circuitous as the one shown in figure 2 decreased to a negligible fraction of all bus routes. If the penalty functions are regarded as a measure of route circuity and route complexity for the resulting minibus routes, a drastic decrease of circuity and complexity can be observed when comparing the penalty values for the minibus routes produced by a run without applying penalties in scoring with another run with applying penalties in scoring. The mean penalty values calculated are much lower for the run with applying penalties (see table 1). Additionally, whereas in the run without applying penalties 329 out of a total of 388 minibus routes would receive some penalty, the share of bus routes receiving some penalty is much smaller for the run with applying penalties at 151 out of 319 minibus routes. However, a few very circuitous minibus routes had very high passenger numbers and could thereby compensate the penalties imposed.

The valueToStartScoring and costFactor parameters shown in table 1 appeared to be a good compromise between reducing circuity and keeping the number of profitable minibus routes and vehicles in operation stable. Keeping all other parameters identical, a run of the Cape Town scenario with the penalties functions gave about 10% fewer minibus vehicles than a run without penalties. For other scenarios it seems reasonable to keep valueToStartScoring at similar levels, but costFactors have to be adapted to the fare and cost rates set for the individual scenarios. Even though it might be tempting to set low values of valueToStartScoring and high costFactors to reduce the circuity even more, in the present setup the penalties also act as a monetary tax and can thereby drastically reduce the number of surviving minibus routes.

5. Conclusion

This paper deals with improvements to the computational speed and to the realism of the produced bus routes of an evolutionary minibus network design process proposed by Neumann [3]. Replacing the default Dijkstra-based public transit router with a RAPTOR implementation and reducing the number of minibus stops inside junction areas speeds up the replanning step dramatically and makes the larger Cape Town scenario computationally feasible.

Before, the generated minibus routes tended to be circuitous and had many side trips and overlapping route segments. Introducing penalties for the route design in minibus plan scoring was highly effective in reducing the high number of circuitous routes. At the same time it reduces the number of surviving profitable minibus vehicles by about 10%. If desired, this effect could be remedied by slightly increasing fare rates or slightly reducing operation cost rates.

References

[1] Delling, D., Pajor, T., Werneck, R.F., 2015. Round-based public transit routing. Transportation Science 49, 591–604. URL: http://dx.doi.org/10.1287/trsc.2014.0534, doi:doi:10.1287/trsc.2014.0534.

[2] Horni, A., Nagel, K., Axhausen, K.W. (Eds.), 2016. The Multi-Agent Transport Simulation MATSim. Ubiquity, London. doi:doi:10.5334/baw.

[3] Neumann, A., 2014. A paratransit-inspired evolutionary process for public transit network design. Ph.D. thesis. TU Berlin. Berlin. arXiv:urn:nbn:de:kobv:83-opus4-53866.

[4] Rieser, M., 2010. Adding Transit to an Agent-Based Transportation Simulation: Concepts and Implementation. Ph.D. thesis. TU Berlin. Berlin. doi:doi:10.14279/depositonce-2581, arXiv:urn:nbn:de:kobv:83-opus-27852.

[5] Rieser, M., Métrailler, D., Lieberherr, J., 2018. Adding Realism and Efficiency to Public Transportation in MATSim, in: 18th Swiss Transport Research Conference, Monte Verità / Ascona, May 16-18, 2018.

[6] SBB, accessed 07-jan-2019. MATSim-Extensions by SBB. https://github.com/SchweizerischeBundesbahnen/matsim-sbb-extensions.

[7] Zhou, C., Frankowski, D., Ludford, P., Shekhar, S., Terveen, L., 2004. Discovering personal gazetteers: An interactive clustering approach, in: Pfoser, D. and Cruz, I.F. and Ronthaler, M. (Ed.), GIS 2004, pp. 266–273. doi:doi:10.1145/1032222.1032261.