Hindawi
Journal of Advanced Transportation
Volume 2021, Article ID 6684795, 18 pages
https://doi.org/10.1155/2021/6684795

Research Article

An Exact Solution Approach for the Bus Line Planning Problem with Integrated Passenger Routing

Evert Vermeir,[1] **Wouter Engelen**,[2] **Johan Philips**,[3] **and Pieter Vansteenwegen**[1]

[1]*KU Leuven, Leuven Mobility Research Centre-CIB, Celestijnenlaan 300, 3001 Leuven, Belgium*
[2]*UC Leuven-Limburg, Research & Expertise Digital Solutions, Geldenaaksebaan 335, 3001 Leuven, Belgium*
[3]*KU Leuven, Division RAM-Flanders Make, Celestijnenlaan 300, 3001 Leuven, Belgium*

Correspondence should be addressed to Evert Vermeir; evert.vermeir@kuleuven.be

Received 16 November 2020; Revised 19 July 2021; Accepted 3 September 2021; Published 4 October 2021

Academic Editor: Gonçalo Correia

Copyright © 2021 Evert Vermeir et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

The bus line planning problem or transit network design problem with integrated passenger routing is a challenging combinatorial problem. Although well-known benchmark instances for this problem have been available for decades, the state of the art lacks optimal solutions for these instances. The branch and bound algorithm, presented in this paper, introduces three novel concepts to determine these optimal solutions: (1) a new line pool generation method based on dominance, (2) the introduction of essential links, i.e., links which can be determined beforehand and must be present in the optimal solution, and (3) a new network representation based on adding only extra edges. Next to presenting the newly obtained optimal solutions, each of the abovementioned concepts is examined in isolation in the experiments, and it is shown that they contribute significantly to the success of the algorithm.

1. Introduction

The design of a public transport network is a multistep process [1, 2]. It starts with the infrastructure network design. Decisions on the physical network, such as bus stops or bus only lanes, are made in this step. The next step is the line planning or the transit network design. In this step, the public transport operator decides where its vehicles will drive and which stops will be served in which order. Then, the timetabling step produces a fixed time schedule for each bus. The setting of frequencies is sometimes included in the line planning step and sometimes in the timetabling step [3]. The next step is to plan the rolling stock and the crews. The final component of the planning process is the dispatching strategy. While all these subproblems influence each other, they are mostly solved sequentially in practice [1]. This paper focuses on the second step, namely, the line planning problem, without considering frequencies. This is also called the uncapacitated line planning problem.

The uncapacitated line planning problem is a difficult combinatorial problem with an enormous search space. There are two main conflicting goals present in nearly every line planning problem. At the one hand, a public transit company wants to spend as little money as possible, and at the other hand, the passengers want to get to their destination as fast as possible. One of the common approaches is to optimize the quality of the service for the passengers (described by the average travel time and/or the number of transfers) and to limit the costs by putting constraints on the number of lines that can be used and the length of the lines (e.g., [4–10]). In order to evaluate the average travel time, the transit assignment, i.e., how the passengers will use the different bus lines to travel through the network, needs to be integrated in the decision process. If frequencies are not considered, an "all-or-nothing" shortest path assignment is typically used [4, 8, 9, 11–15]. Mandl's Swiss network [14], is the only widely used benchmark network for the uncapacitated line planning problem [4, 7–10, 12, 13, 15–21]. Mandl's network with 15 nodes and 21 links is illustrated in Figure 1. The length of each link is indicated with the number next to the link. Benchmark instances are available with two different maximum line lengths and with different

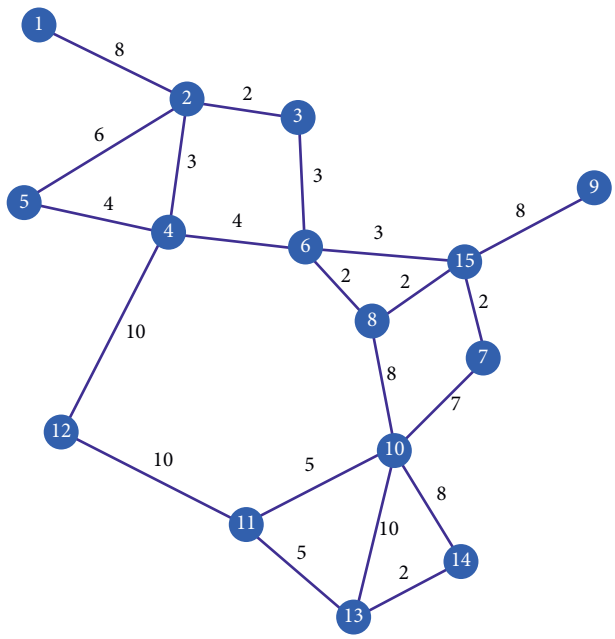

Figure 1: Mandl's Swiss network [14].

numbers of lines. Despite it being a small network with only fifteen nodes, there are no known optimal solutions for instances with more than three bus lines on this network [22].

The main contribution of this paper is that, for the first time, all these benchmark instances for the uncapacitated line planning problem with integrated passenger routing are solved to optimality. The amount of lines necessary to offer each passenger a direct connection along his/her shortest path between origin and destination is also determined for the first time. Furthermore, a number of novel concepts are introduced which are required to obtain these optimal solutions, but which are also interesting for future research on this complex planning problem.

First, a new method is developed to construct a line pool of all feasible lines that can be present in an end solution. Secondly, the concept of "essential links" is presented. These are parts of lines, determined during the branch and bound process, which must be part of the optimal solution. Thirdly, the "Direct Link Network" representation (DLN) is introduced. This is a novel network representation allowing a faster evaluation of solutions, compared to the well-known change-and-go network representation [23]. Finally, a new branch and bound method limits the amount of required evaluations before obtaining the optimal solution. The method branches over all possible lines containing a certain Origin-Destination (OD) pair. The experimental results illustrate the importance of each of these concepts. We also make clear that research on the line planning problem should shift to larger and more realistic instances.

The paper is organized as follows. Section 2 contains an overview of the line planning problem literature, focusing on (meta) heuristics and exact approaches for the line planning problem with integrated passenger routing. Section 3 describes in detail which variant of the line planning problem is tackled in this paper. Section 4 discusses the solution approach. It shows the branch and bound algorithm and explains all the details about the essential links and the Direct Link Network representation. Section 5 contains the results of the experiments on Mandl's Swiss Network [14]. The paper ends in Section 6 with the conclusions.

2. Literature Review

This section contains a short survey of the literature on the line planning problem, with and without frequency setting. Both (meta) heuristics and exact approaches will be covered. A more extensive overview can be found in the following papers and books: [1, 3, 22, 24–27]. It should be noted that, depending on the community, this planning problem is called the transit network design problem or the line planning problem. In this paper, the term line planning will be used.

2.1. Heuristics and Metaheuristics. Early research on different variants of the line planning problem focused on heuristics. One of the earliest heuristics to solve the line planning was by Patz [28]. In this work, lines are iteratively removed from a large set of lines based on a penalty structure. Mandl [29] combined both passenger demand and shortest paths to create an initial solution. Then, this solution is iteratively improved aiming to optimize the total travel time. These early heuristics were unable to solve large instances, but they provide the basis for the development of the more modern metaheuristics [11].

The first metaheuristics for solving the line planning problem were Genetic Algorithms (GA). Chakroborty and Wivedi [30] and Pattnaik et al. [31] were some of the first to solve the line planning problem with a GA. Because they were one of the first ones to use metaheuristics during the optimization process, the performance of their algorithms was a lot better than the previously known methods. Since then, many refinements have happened to these GA's. Zhao and Zeng [32] added a local search component to the GA, resulting in a memetic algorithm. Nayeem et al. [8] introduced elitism in their GA as well as a guided local search. Islam et al. [7] used a stochastic beam search to tackle the line planning problem. One of their other contributions is the development of a new heuristic to get a very strong starting solution. Their method combines edge lengths with demand served into a single cost, while previous methods just tried to serve a high demand along (quasi) shortest paths. Chai and Liang's [33] work is an example of a recent paper using a GA, and they developed a modified version of the well-known NSGA-II algorithm to solve the line planning problem. Fan and Mumford [12] used both hill climbing and simulated annealing. Also swarm intelligence techniques are being used in line planning such as Blum and Mathew [5] who use Ant Colony Optimization or Nikolić and Teodorović [9] who use Bee Colony Optimization. Vermeir et al. [34] use an iterated local search combined with a local evaluation method which only looks at a cut of the network to quickly evaluate candidate solutions.

Mauttone and Urquhart [35] were the first ones to model the line planning problem as a multiobjective problem. They used a greedy randomized adaptive search procedure to generate a Pareto front with minimal values for operator and passenger cost. Recently, Duran-Micco et al. [6] developed a memetic algorithm to add emissions as an extra objective. They showed that emissions can be greatly reduced while only having a minimal impact on the travel time and operator costs. Research focusing only on minimizing the operator cost is not discussed here, but the interested reader is referred to the above mentioned survey papers or e.g., [17].

2.2. Exact Approaches. A number of publications on exact solution approaches start by defining a mathematical model and then try to solve the problem or a relaxation of the problem. As illustrated by the paper below, most of these line planning models originate from train line planning. Since trains typically have lower frequencies than urban bus networks, the selected frequencies have a much larger impact on the transfer times than for a high frequency network. Capacity limitations at stops (stations) are more important for rail systems and are typically checked through constraints based on frequencies [27]. Hence, most commonly, the setting of frequencies is included in the line planning process [36]. While frequencies should have a direct relation to transfer times, including frequency-based transfer times lead to non-linear optimization models [36]. Therefore, even when including frequency setting, typically transfer times are considered through a transfer penalty [23, 37–40]. These line planning problems are typically formulated as a (mixed-) integer program.

There exist many ways to calculate passenger routing in line planning. Distributing all the passengers along their shortest path is the most commonly used strategy in line planning. While this is a simple method, it is already computationally expensive to execute during an optimization [40]. In cost-oriented models the operational cost is minimized subject to certain constraints on the service level. These constraints are typically very easy to check and avoid the expensive calculations. Since costs are easy to calculate, it is possible to use exact approaches even on real-world data. Claessens et al. [41] use a branch and bound procedure to obtain results for the Dutch railway system. Bussieck [42] used this work as a basis to make a cut and branch algorithm which was used to obtain results for both the German and Dutch railway system. Goossens et al. [43] built further on this work with a branch and cut approach which allowed to loosen some constraints regarding the lines. Bussieck et al. [44] created a fast procedure that obtains good solutions in a small computation time. Canca et al. [45] managed to integrate several other planning steps in the optimization process.

When looking at passenger-oriented models, explicit passenger routing is very important and cannot be easily avoided. In these models, an evolution of the objective function can be observed, which directly influences the speed at which the service level can be calculated. The earliest work circumvents the expensive passenger routing by maximizing the amount of direct travelers [46]. Here, the integer program is solved with a branch and bound approach which greedily chooses lines with maximal current direct travelers. Bussieck et al. [47] and Bussieck [42] expanded this work by taking the vehicle capacities into consideration through introducing constraints on the edge frequencies. The repeated passenger calculations can also be avoided through a so-called "system split," where links are categorized by speed. Passengers are then assumed to switch to the fast levels as soon as possible and leave it as late as possible. With these assumptions, the passenger flows can be estimated beforehand and do not have to be recalculated during the optimization process. This method was promoted by Bouma and Oltrogge [48], where it is used in a branch-and-bound approach applied to the Dutch railway network. Later, the objective function evolved to the minimization of the travel time of the passengers [23, 37–40].

More recent research integrates passenger routing in the line planning problem. In order to still model the line planning problem as MIPs, assumptions about passenger routing are made, such as passengers choosing minimal in vehicle travel-time routes (and thus not considering transfers) or routes with a minimal number of transfers [40, 49, 50] or such as passenger routes that can be assigned by the public transport operator [23, 37, 40]. To avoid these limitations, a bilevel model (like in most metaheuristics) can be used. In this bilevel model, network decisions are made at an upper level and the passenger routing decisions are made at the lower level [36, 51, 52]. An early implementation of this is formulated by Constantin and Florian [52]. Goerigk and Schmidt [36] use this bilevel model to completely integrate passenger routing and solve, to optimality, instances with up to ten bus nodes and a line pool of 30 lines randomly generated beforehand. For larger networks, they use a genetic algorithm. However, none of these methods have solved the instances on Mandl's Swiss Network [14] (fifteen nodes) with integrated passenger routing to optimality.

2.3. Benchmark Instances. Despite being too small to be representative of a city bus network, Mandl's Swiss Network is the most commonly used benchmark instance in bus line planning [22]. Nevertheless, fair comparisons in line planning research remains an issue since different variants of the problem are considered [4]. There are a lot of real-life instances used in literature, but they each have very specific objectives and constraints or the data is not publicly available. These issues are also mentioned in Ceder [1], Ibarra-Rojas et al. [25], Farahani et al. [24], Schöbel [27], Kepaptsoglou and Karlaftis [26], and Guihaire and Hao [3]. Fortunately, Mumford [15] recently made four larger datasets publicly available. These networks are based on actual bus route networks from Chinese and British cities and are being used more and more [4, 6, 8, 18–20, 53]. One of the datasets was already used earlier by Fan and Mumford [12], and another one was already used by Nikolić and Teodorović [9] with different parameters.

3. Problem Description

The available infrastructure network is depicted by a directed graph $G = \{V, E\}$, which contains vertices $V = \{v_1, v_2, \ldots, v_n\}$ representing the bus stops and edges $E = \{e_{ij}, e_{kl}, \ldots, e_{yz}\}$ which are the connections available between these bus stops. The cost or travel time on an edge e_{ij} is indicated by t_e. The demand of the passengers traveling through the public transport network is represented by an Origin Destination (OD) matrix D. The number of passengers per hour that want to travel from bus stop i to bus stop j is then depicted by d_{ij}. In the line planning problem, the goal is to select the best possible set L of bus lines. In the uncapacitated line planning problem with integrated passenger routing, the objective function is to minimize the total travel time (TTT) of all passengers. This means the sum of all travel times of all passengers. This travel time also includes transfer times. This time spent waiting on the next bus (and the discomfort of transferring) is modelled with a transfer penalty TP, which penalizes each transfer. It should be noted that a timetable or frequencies are not available at this stage, and therefore, a more accurate modelling is not feasible and waiting times are not considered. In order to evaluate and minimize the TTT, the routing each passenger will take has to be known. This routing results in a set $\pi_{ij} = \{e_{ik}, e_{kl}, \ldots, e_{xj}\}$ of edges used and the amount of necessary transfers τ_{ij}. The total travel time of all passengers TTT is then represented by

$$\mathrm{TTT}(L) = \sum_{i \in V} \sum_{j \in V} \left(d_{ij} \left(\sum_{e \in \pi_{ij}} t_e + \mathrm{TP} * \tau_{ij} \right) \right). \quad (1)$$

To completely integrate the passenger routing in the optimization, the line planning problem is formulated as a bilevel problem as mentioned in the literature review. The design of the lines is the upper problem, and the routing of the passengers is the lower problem. For the lower problem, the "transit assignment," a shortest path allocation is used. It is assumed that each passenger will travel along its shortest path, considering both the in-vehicle travel time and the transfer penalties. Although more complex and accurate transit assignment methods are available [54–57], most line planning problems are still solved making this assumption [4, 7–13, 16, 29, 53]. When operator costs are used as an objective and/or frequencies are considered, frequency-based assignment models are regularly used (e.g., [58–61]).

The objective and constraints used in this paper are the most commonly used in literature for the uncapacitated line planning problem. This is required to allow a fair comparison with the state-of-the-art algorithms. First of all, as mentioned above, a passenger-oriented objective function is used. Therefore, the operator cost is limited by imposing constraints. There is a maximum number of nodes that can be present in each line, and there is a maximum number of lines that can be selected in the line plan. The shape of the bus lines is only limited by not allowing any stop to be visited twice. This excludes all loops in a given line. This also means that, in this paper, the set of feasible lines is not limited or fixed beforehand, as is the case in some other papers (e.g., [36, 37, 51, 62]). When a line is selected, it is assumed to be served in both directions and the bus capacity is assumed to be high enough to serve all passengers. The integration of the passenger routing and the fact that no limited set of lines is considered are two aspects significantly complicating the uncapacitated line planning problem considered in this paper.

4. Methodology

This section starts with a detailed discussion of a number of essential concepts and components implemented in our algorithm to optimally solve the uncapacitated line planning problem with integrated passenger routing. First, it is explained how the line pool is generated and how "essential links" can be determined. Then, a new transit network representation is constructed, and it is shown how it can be used to efficiently calculate the passenger routing. Finally, the actual branch and bound algorithm is explained in detail. In Section 4.6, a summary of the method is given together with the pseudocode and a flowchart, and two alternative approaches are briefly introduced.

4.1. Line Pool Generation. To ensure that the entire search space is explored, a pool of all possible lines is generated. To construct this pool, an important property of the problem is exploited. The total travel time of all passengers will never increase when a line is extended by adding an extra stop at one of the ends. Therefore, all lines that are a subline of another feasible line are dominated by that line. When a dominated line would be part of the optimal solution, it can be substituted for its dominator. This means that only considering nondominated lines will be sufficient to find the optimal solution. This is illustrated in Figure 2, where line (a) is dominated by line (b). All passengers that use line (a) in their shortest path will still be able to use the same shortest path if we transform line (a) to line (b). The situation can only improve for the passengers. For example, passengers that want to travel from node 5 to node 3 now have a direct connection along the shortest path. Because the uncapacitated line problem does not consider frequencies or costs, line (b) is always at least as good as line (a). This means that we can guarantee to obtain an optimal solution without having to consider line (a). This is the power of domination, and it significantly limits the size of the line pool. For example, on Mandl's network with infinite line lengths, this results in 581 lines that could be present in the final solution, while without the dominance rules, there would be 8180 candidate lines. Note that this dominance rule only holds for the uncapacitated line planning problem. If frequencies are included or when demand elasticity is considered the method will have to be adjusted.

A pool of nondominated lines is now constructed recursively. The algorithm starts by selecting any node in the network and connecting one of its adjacent nodes to construct a bus line. This bus line is extended by adding new adjacent nodes until it is no longer possible without violating any constraints (maximum line length or visiting a stop twice). A line that cannot be extended is a possible candidate for the optimal solution and is added to the line pool. After

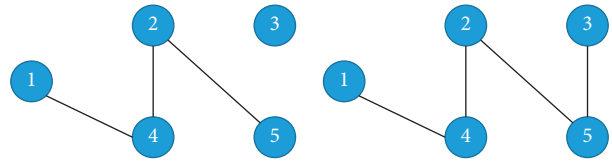

Figure 2: Example of domination. Line (a) is dominated by line (b).

undoing the last extension, a different adjacent node can be chosen for extending the line. When there are no adjacent nodes left, the previous extension is undone, and so on. This process continues until only the starting node remains. If this is repeated for all nodes in the network, a pool of all possible lines is generated. Since all lines are considered bidirectionally, symmetric lines can be eliminated.

4.2. Essential Links. An essential link is a link for which it can be determined beforehand that it has to be present in the optimal solution. All candidate solutions that do not contain all essential links do not have to be evaluated. This results in fewer candidate solutions that have to undergo a time-consuming evaluation. It will be shown in the experimental results (Section 5) that this makes the algorithm significantly faster.

Essential links are determined by removing a link from the infrastructure network and then solving the all pair shortest path problem. This assumes every passenger will have a direct connection along its shortest path, but the removed link cannot be used. If the total travel time of all passengers obtained this way is worse than the best-known solution, then the removed link is an essential link. In other words, a lower bound is calculated for the network where a certain link is removed. If this lower bound is higher than an upper bound that is already available at the start of the algorithm (the best-known solution), then the removed link is essential. Without this link, the optimal solution can never be obtained. After repeating this process for all links in the network, a list of all essential links is constructed.

This paper uses the optimal solution with one less line as the best-known solution to determine the essential links. For example, when starting to solve the problem for six lines, the optimal solution for five lines is used as the best-known solution to determine the essential links for six lines. Results known from literature or (meta) heuristics could also be used as the best-known solution. If a problem currently has no known solution, any solution method can be used to get a first upper bound. Obviously, the better the quality of this upper bound, the more essential links can be identified. Note that the lower bound can be obtained when not including a link is constant and can be precalculated. This means that the list of essential links could be updated on the go every time a new best solution is found. However, in our algorithm, the list of essential links is constructed during a precalculation phase. The presence of each of the essential links is also precalculated for each line in the generated line pool. This makes checking these added constraints during optimization very fast.

4.3. Direct Link Network Representation. To take transfers into account when looking for the passenger routes, the available infrastructure network needs to be extended to obtain a proper representation. Typically, this is done by adding a dummy node for every stop on every bus line. This type of extended network is also called the Change and Go Network (CNG) [23, 36] or the Train Service Network (TSN) [63]. A disadvantage of this method is that the addition of many extra nodes significantly impacts the time required to calculate the passenger routes.

This paper does not use the TSN or CNG. Rather than adding extra nodes, extra links are added to the network. For any two nodes that are connected by a single bus line, a direct link is added to the network. The total travel time of the bus between these two nodes is then used as travel time for this link. If multiple bus lines connect the same nodes, only the link with the shortest travel time is kept in the final network, which we call the "Direct Link Network (DLN)."

Figure 3 illustrates both the CNG and the DLN on a small toy network. Figure 3(a) depicts a small toy network with five stops and two bus lines: a full black line and a dotted orange line. Figure 3(b) is the DLN representation of the same toy network. Three extra links have been added to the network. The links in the DLN are color coded to make it clear from which line each link originates. Node one has a direct connection to every other node in the network; hence, it has a link to every other node in the network. But node one can reach node two with a direct connection through each of the two lines. Because the connection through the orange line is shorter, this is the only link that is kept in the representation. Node three and node five are not connected through a direct connection. Thus, passengers traveling between these nodes need to use a transfer. In the DLN, this is represented by the absence of a link connecting the two nodes. If a shortest path is calculated on the DLN, then each link beyond the first that is part of the shortest path also represents a transfer (and comes with a penalty). Figure 3(c) is the CNG representation of the toy network. Seven nodes and seven links have been added to the network.

There are two ways to consider transfers in this DLN. The simplest way is to add the transfer penalty to the length of every link in the network. Any method to solve the all pair shortest path problem can then be used to calculate the passenger routes. Since every link now contains a transfer penalty, a single transfer penalty has to be subtracted from each shortest path calculated this way. Because of the nature of the DLN, every additional link used beyond the first implies an actual transfer (for which the penalty is indeed included). However, we decided to incorporate the transfer

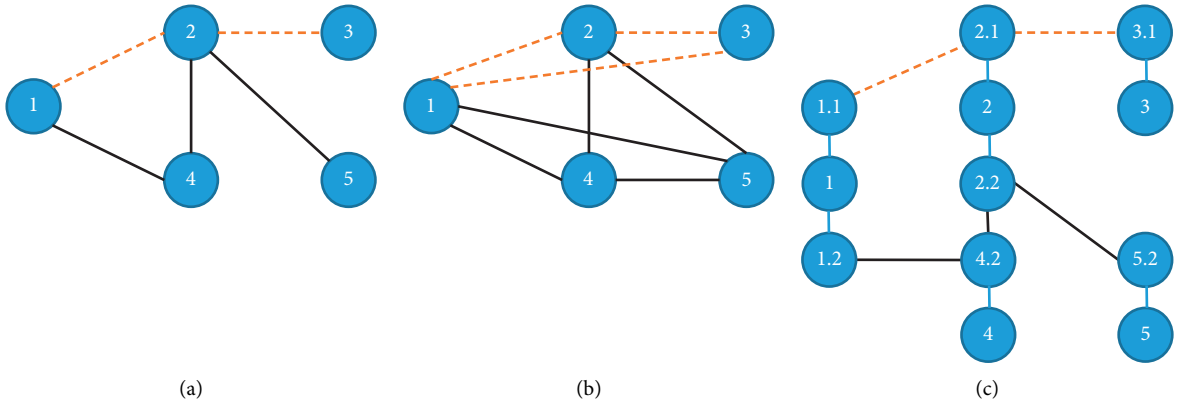

Figure 3: (a) Toy network with 2 bus lines; (b) Direct Link Network representation. (c) Change and Go representation.

penalties by slightly modifying Floyd–Warshall's algorithm, which is very simple for the DLN representation proposed in this paper. Direct connections in the network are represented by a single link. Therefore, every combination of links implies a transfer. Then, a transfer penalty can simply be added to the main operator of the Floyd–Warshall algorithm, which is illustrated in

$$\forall k, i, j\colon \operatorname{dist}(i, j) = \min\left(\operatorname{dist}[i][j], \operatorname{dist}[i][k] + \operatorname{dist}[k][j] + \mathrm{TP}\right). \tag{2}$$

Also, in Dijkstra's algorithm, the transfer penalties can be incorporated directly. Actually, Dijkstra's algorithm is most commonly used in line planning research. This paper, however, uses Floyd–Warshall's algorithm since it performed better in the initial testing. Note that the method used in this paper creates a much more dense network than the traditional CNG network. Floyd–Warshall tends to perform better on dense networks, while Dijkstra tends to do better on sparse networks [64, 65].

It should be noted as well that this DLN representation can also be used in line planning research using metaheuristics. It can be especially useful when a high number of lines are considered. This network can also be adjusted easily to work for line planning with frequencies. Whenever a single OD pair is connected by multiple bus lines, this would also result in multiple links, one for each bus line available between these nodes, instead of only keeping the shortest one as explained above.

4.4. Branch and Bound. At the start of the algorithm, the lower bound corresponds to the ideal situation. This would mean that every passenger is able to travel from its origin to its destination along the shortest possible path in the infrastructure network without any transfers. Then, as illustrated in Figure 4, the branching process starts by selecting the first OD pair (OD1) from the sorted list of OD pairs. The way these OD pairs are sorted is explained in Section 4.5. In the explanation below, we assume that OD1 has node A as origin and node B as destination. From the pool of lines, all lines that contain both node A and node B are selected (OD1 L1, OD1 L2, etc.). In this selection, only those lines for which the detour (compared to the shortest possible path) between A and B is less than a single transfer penalty are considered as branches in the branch and bound tree. These lines are then sorted according to their travel time between A and B. Now, two different scenarios are possible. One of these lines is chosen as part of the solution, each leading to a different branch (OD1 L1 is part of the solution or OD1 L2 is part of the solution), or none of these lines is chosen, leading to one additional branch (≥TP in Figure 4).

When a line is chosen as a part of the solution, this implies that, in the end solution, all passengers traveling from A to B will travel along this line. This is due to the fact that shorter alternatives for traveling from A to B have been considered in previous branches. For instance, if OD1 L2 is selected as part of the solution, OD1 L1 was not selected and the shortest path to travel from A to B in the solution will be along OD1 L2. This path is then called the "optimal path" between A and B in this branch. This has an important implication. If the optimal path is longer than the "shortest possible path" between A and B, then the lower bound for this branch can be adjusted with the difference between the "optimal path" and the "shortest possible path," multiplied with all the demand between A and B. Obviously, if this makes the lower bound of this branch worse than the current upper bound available, this branch can be pruned. There is also an effect for all OD pairs lying on the part of the chosen line between A and B. Since they contribute to the "optimal path" between A and B, their own "optimal path" cannot improve the connection between A and B. This is also taken into account in the lower bound.

If none of these lines (OD1 L1, OD1 L2, etc.) is chosen, the additional branch is followed and the "optimal path" between A and B in that branch will be at least a single transfer penalty longer than the "shortest possible path." In Figure 4, this is represented by the branches called "≥TP." In the best-case scenario in this branch, traveling from A to B is possible along the shortest possible path with a single transfer. Therefore, in this branch, the lower bound can be adjusted by adding a single transfer penalty multiplied with the total demand between A and B. The new lower bound has to be compared to the current upper bound to decide whether to continue along this branch or not.

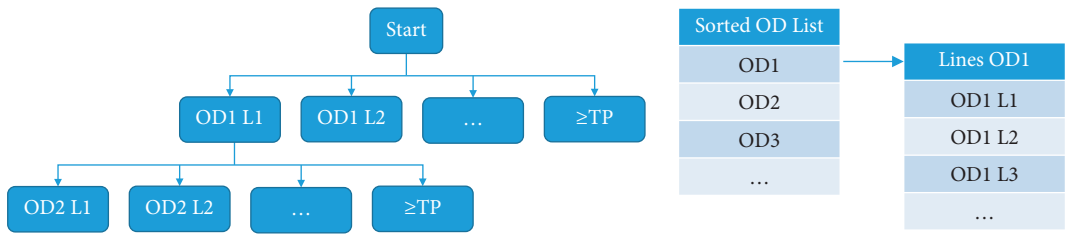

Figure 4: The branching process.

In the next branching step, the next OD pair from the sorted list is chosen (OD2) and the process described above is repeated for this OD pair. Again, all lines from the line pool with an optimal path that deviates less than a single transfer penalty from the minimum possible are selected to branch upon, together with one branch were none of these lines is selected. There is one difference however. All candidate lines with an optimal path for OD1 that is shorter than the chosen one should not be considered again and thus cannot be branched upon. Since the lines were sorted by length (or travel time), all previously branched upon lines are excluded from being further explored. In Figure 4, this means that if one of the possible branches of OD2 is a line already explored in an upper branch, it is immediately pruned. If, for example, OD1 L2 is the current branch being explored, then the lines OD1 L1 and OD1 L2 will not be considered as lines for OD2. But OD1 L3 could be a valid candidate since it has not been explored before.

This branching process continues either until the new lower bound is worse than the current upper bound, after which the branch gets pruned, or until the required number of lines is chosen. After sufficient lines are chosen, the presence of all nodes and essential links is checked. If they are all present, the solution is evaluated and the upper bound is adjusted if a new best solution is found.

4.5. Sorting of OD Pairs. The order in which OD pairs are selected to be branched upon greatly impacts the calculation time of the algorithm. There are two elements considered when selecting the next OD pair. One is to have as little lines as possible with a detour smaller than the transfer penalty. The other is to have a large demand for an OD pair. The first limits the number of branches that need to be constructed, and the latter increases the lower bound faster, allowing to prune more frequently. In this paper, experimental results for three different sorting methods for OD pairs are discussed. The main sorting method combines both elements mentioned earlier in this paragraph by first sorting the OD pairs by decreasing the number of nodes in between and then breaking any ties by putting the highest demand first. This is the main sorting method used in the experiments. It makes sure that the top levels have as little branches as possible.

A first alternative sorting method, which will be called "complex sort," combines the number of branches and the demand in a single variable. The number of branches is equal to the amount of lines that contain the OD pair under consideration and that do not make a detour of at least one transfer penalty compared to the shortest path (plus one extra branch for where the detour is assumed to be at least a transfer penalty, and thus, no line is selected). The complex sort calculates the total amount of branches and divides this by the square root of the demand of the chosen OD pair to branch upon. The square root is used to add more weight to the number of branches compared to the size of the demand. In this way, the algorithm tries to limit the number of top branches, while also maximizing the impact of not selecting a line. Finally, the second alternative sorting method is a random sort. This is used to prove whether the sorting actually has an effect on the performance of the algorithm.

4.6. Summary of the Exact Algorithm. Figure 5 shows an overview of the entire algorithm in a flowchart, and Algorithm 1 presents the pseudocode. The algorithm starts with some precalculations. The most important precalculation is the generation of the full set of bus lines that could end up in an optimal solution (line 2 in Algorithm 1). All origin-destination (OD) pairs are sorted by the number of nodes along the shortest path and their demand (line 3 in Algorithm 1). All the branches for each OD pair are also determined in this step, so it is checked for each OD pair which lines contain that OD pair and how long the detour is compared to the shortest path (line 4 in Algorithm 1). Then, the essential links are calculated (line 5 in Algorithm 1). Finally, the initial lower bound corresponds to the (probably unfeasible) solution where every passenger travels along the shortest possible path in the infrastructure network without any transfers. Based on all these precalculations, the branch and bound algorithm can commence (line 6 in Algorithm 1).

The branch and bound algorithm branches on all possible lines from the line pool that connects a chosen OD pair with a detour smaller than a transfer penalty (line 16 in Algorithm 1). One additional branch is considered where none of these lines are allowed (line 24 in Algorithm 1). Here, the lower bound is increased with the transfer penalty multiplied with the demand of the OD pair (line 25 in Algorithm 1). The OD pair to branch on next is always selected based on the precalculations. The branch and bound algorithm keeps branching deeper and deeper until the set amount of bus lines is selected (line 7 in Algorithm 1) or until the lower bound of a branch exceeds the current best solution (line 20 in Algorithm 1). Whenever the set amount of lines is selected, the presence of all nodes and essential links is checked (line 8 in Algorithm 1). If this is the case, the Direct Link Network is constructed and the passenger routes are determined using the adapted Floyd–Warshall

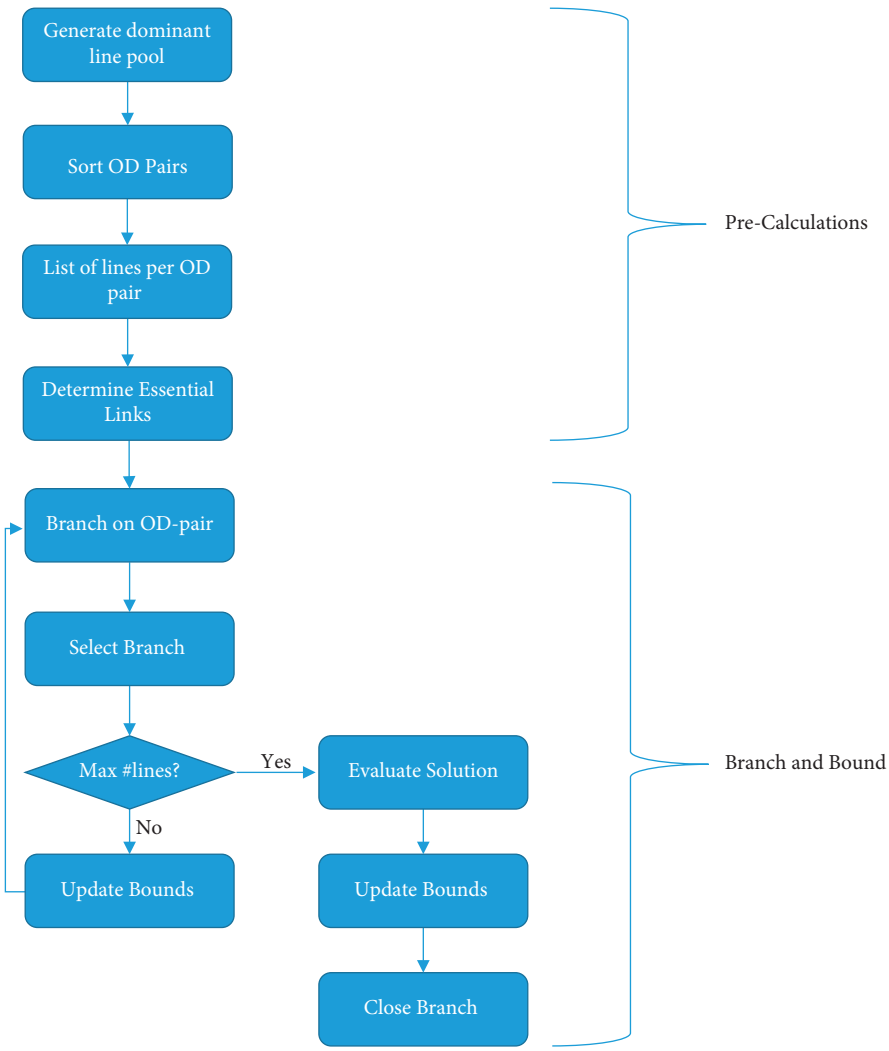

Figure 5: Flowchart of the algorithm.

algorithm. The solution is then evaluated and compared to the current upper and lower bound. At the end, all relevant parts of the search space will have been explored and the optimal solution is determined.

4.7. Alternative Approaches. The most obvious approach to find an optimal solution is a simple brute force solution. By selecting all possible combinations of n lines out of the feasible line pool, all feasible solutions can be evaluated. The computation time of this approach increases drastically for each extra line that can be part of the solution. This approach becomes intractable in even very small instances, such as Mandl's network with five lines.

Preliminary experiments on the Mumford0 instance with 30 nodes and 90 links show that it takes up too much memory to calculate all possible lines beforehand. To address this, the line pool can be generated on the go. After selecting an OD pair to branch on, the pool of lines with a detour of less than one transfer penalty can be calculated. Initial experiments on larger networks and with a higher maximum number of lines took too much time to even come close to the results found by heuristics and are, hence, not included in the results. To use the insights gained in this paper for larger networks, extra adjustments have to be made. This will be discussed further in Section 6.

5. Results

This section contains the results of the experiments on Mandl's Swiss Network [14]. First Mandl's Swiss Network is introduced. The actual experiments start with determining, for the first time, the optimal solutions for all available instances on the network with the exact algorithm discussed above. These solutions are also compared to the solutions found by state-of-the-art (meta) heuristics. Then, the importance of the newly introduced concepts is analyzed: the sorting method used, the essential links, the Direct Link Network, and finally using the entire branch and bound method instead of brute forcing a solution.

The software algorithms were implemented in C++17 and compiled with g++ (GCC) 9.3.0, and Docker was used to setup a standalone image to run the experiments. The Docker containers were executed on a dedicated virtual

```
(1) Precalculations:
(2)    Generate pool of dominant lines (Section 4.1);
(3)    Sort OD Pairs (Section 4.5);
(4)    Construct list of lines per OD pair (Section 4.5);
(5)    Determine Essential Links (Section 4.2);
(6) Recursive Branch and Bound (Section 4.4):
(7)    If (size of Current_Line_Plan equals maximum)
(8)       If (All essential links and nodes are present in Current_Line_Plan)
(9)          If (TTT < UpperBound)
(10)            Optimal_Line_Plan ⟵ Current_Line_Plan;
(11)            UpperBound ⟵ TTT;
(12)         end
(13)      end
(14)   Else
(15)      Select OD pair
(16)      For (All lines serving the OD pair with a detour less than TP)
(17)         Next_Line_Plan ⟵ Current_Line_Plan;
(18)         Add line to the Next_Line_Plan;
(19)         Next_LowerBound ⟵ LowerBound + detour ∗ Demand of OD pair;
(20)         IF (Next_LowerBound < UpperBound)
(21)            Go one step deeper in the branch and bound with Next_Line_Plan
(22)         end
(23)      end
(24)      Go one step deeper in branch and bound without selecting a line (deviation > TP)
(25)      LowerBound ⟵ LowerBound + TP ∗ Demand of OD pair;
(26)   end
(27) Return Optimal_Line_Plan
```

Algorithm 1: Pseudocode of the algorithm.

machine, running CoreOS, to minimize context switching and external interference. The virtual machine ran on Intel(R) Xeon(R) CPU E5-2640 v4@2.40 GHz hardware and was granted 4 dedicated CPU cores (8 HyperThreads) and 16 GB dedicated RAM. The C++ implementation exploits the multicore setup by parallelizing the execution on several worker threads. All the data of the experiments and the instances is available at https://www.mech.kuleuven.be/en/cib/lp/mainpage#section-12.

5.1. Mandl's Swiss Network. Mandl's Swiss Network (Figure 1) is a small network with 15 nodes and 21 links originally used in Mandl [14]. In total, there are 15570 trips in the network and the demand is symmetric. It is one of the only publicly available datasets for the line planning problem and, therefore, the most used benchmark instance. The most commonly used parameters for this network are used in this work. The most common limiter of line length is by limiting the number of nodes per line to eight. In this paper, both a maximum of eight nodes per line and an unlimited number of nodes per line are used. These "unlimited" lines will then be limited by the fact that nodes can only be included once in a single line. For finding the optimal solutions, the results are shown for all instances with a relevant amount of lines. When the newly introduced concepts are analyzed, only a subset of instances will be used to somewhat limit the required calculation time. The transfer penalty is set to five minutes, and this value is used in most publications [4, 7, 8, 10].

5.2. Exact Solution Algorithm. The algorithm used here works entirely as described in Section 4.6. Table 1 shows the optimal solutions with respect to Total Travel Time for all instances (Number of Lines) with at most eight nodes per line. The third column shows the Total Travel Time (TTT) and the fourth column the Average Travel Time (ATT) in minutes, which is the TTT per trip. The fifth column shows the CPU time of the algorithm, in seconds. This CPU time is the time the algorithm spent in the branch and bound procedure. The entire precalculations require close to 40 seconds (of which nearly all time is spent preparing the branches), but does not have to be repeated for every instance. The final column shows the number of solutions that were evaluated, in millions. Table 2 shows the actual lines that make up the optimal line plan for each instance. As a visual example, Figure 6 represents the optimal solution for the instance with four lines and at most eight stops per line. Note that every line passes by node ten. This is expected since this node alone is responsible for more or less one quarter of all the demand in the network.

The CPU time keeps increasing until nine lines. Every extra line increases the amount of possible solutions drastically. Therefore, it is expected that more lines result in longer CPU times and more solutions checked. But, at ten lines, the search time decreases again. There are several effects at play here. More lines make it easier for the algorithm to get good upper bounds quickly because more lines result in better solutions. These upper bounds will also be closer to the absolute lower bound. Both of these things

Table 1: Optimal solutions for Mandl's Swiss network with at most 8 nodes per line.

Number of lines	Max nodes per line	TTT (min)	ATT (min)	CPU time (s)	Solutions checked (M)
3	8	169450	10.88	13	<1
4		163210	10.48	297	15
5		160450	10.31	3317	466
6		158500	10.18	18856	3785
7		157260	10.10	39753	6737
8		156750	10.07	161902	25704
9		156300	10.04	200872	24934
10		155940	10.02	29296	4032
11		155850	10.01	17054	1369
12		155820	10.01	16320	1044
13		155800	10.01	24883	1634
14		155790	10.01	1019	177

Table 2: Optimal solutions for 3 to 14 lines with at most 8 nodes per line.

3 lines	4 lines	5 lines
Line 1: 1, 2, 3, 6, 8, 10, 11, and 13	Line 1: 1, 2, 3, 6, 8, 10, 11, and 12	Line 1: 1, 2, 3, 6, 8, 10, 14, and 13
Line 2: 5, 4, 6, 15, 7, 10, 14, and 13	Line 2: 1, 2, 5, 4, 6, 8, 10, and 11	Line 2: 1, 2, 3, 6, 15, 7, 10, and 11
Line 3: 2, 4, 12, 11, 10, 8, 15, and 9	Line 3: 10, 7, 15, 6, 3, 2, 4, and 12	Line 3: 1, 2, 5, 4, 6, 8, 10, and 11
	Line 4: 9, 15, 8, 10, 14, 13, 11, and 12	Line 4: 2, 4, 12, 11, 10, 7, 15, and 9
		Line 5: 7, 15, 8, 6, 4, 12, 11, and 13
6 lines	**7 lines**	**8 lines**
Line 1: 1, 2, 3, 6, 8, 10, 11, and 13	Line 1: 1, 2, 3, 6, 8, 10, 11, and 13	Line 1: 1, 2, 3, 6, 8, 10, 11, and 13
Line 2: 1, 2, 3, 6, 15, 7, 10, and 14	Line 2: 1, 2, 3, 6, 15, 7, 10, and 14	Line 2: 1, 2, 3, 6, 15, 7, 10, and 14
Line 3: 1, 2, 5, 4, 6, 8, 10, and 14	Line 3: 1, 2, 5, 4, 6, 8, 10, and 14	Line 3: 1, 2, 5, 4, 6, 8, 10, and 14
Line 4: 3, 2, 5, 4, 6, 8, 15, and 7	Line 4: 3, 2, 5, 4, 6, 8, 15, and 7	Line 4: 3, 2, 5, 4, 6, 15, 7, and 10
Line 5: 5, 4, 12, 11, 10, 7, 15, and 9	Line 5: 5, 4, 12, 11, 10, 7, 15, and 9	Line 5: 5, 4, 12, 11, 10, 7, 15, and 9
Line 6: 1, 2, 4, 12, 11, 13, 14, and 10	Line 6: 9, 15, 6, 3, 2, 4, 12, and 11	Line 6: 9, 15, 6, 3, 2, 4, 12, and 11
	Line 7: 1, 2, 4, 12, 11, 13, 14, and 10	Line 7: 1, 2, 4, 12, 11, 13, 14, and 10
		Line 8: 12, 4, 6, 8, 15, 7, 10, and 13
9 lines	**10 lines**	**11 lines**
Line 1: 1, 2, 3, 6, 8, 10, 11, and 13	Line 1: 1, 2, 3, 6, 8, 10, 11, and 13	Line 1: 1, 2, 3, 6, 8, 10, 11, and 13
Line 2: 1, 2, 3, 6, 15, 7, 10, and 13	Line 2: 1, 2, 3, 6, 15, 7, 10, and 8	Line 2: 1, 2, 3, 6, 15, 7, 10, and 8
Line 3: 1, 2, 3, 6, 15, and 9	Line 3: 1, 2, 3, 6, 15, and 9	Line 3: 1, 2, 3, 6, 15, and 9
Line 4: 12, 4, 2, 3, 6, 8, 10, and 14	Line 4: 12, 4, 2, 3, 6, 8, 10, and 14	Line 4: 12, 4, 2, 3, 6, 8, 10, and 14
Line 5: 1, 2, 5, 4, 6, 8, 10, and 11	Line 5: 5, 4, 6, 8, 10, 11, 13, and 14	Line 5: 5, 4, 6, 8, 10, 11, 13, and 14
Line 6: 3, 2, 5, 4, 6, 8, 15, and 7	Line 6: 1, 2, 5, 4, 6, 15, 7, and 10	Line 6: 1, 2, 5, 4, 6, 15, 7, and 10
Line 7: 5, 4, 12, 11, 10, 7, 15, and 9	Line 7: 5, 4, 12, 11, 10, 7, 15, and 9	Line 7: 5, 4, 12, 11, 10, 7, 15, and 9
Line 8: 9, 15, 8, 6, 4, 12, 11, and 10	Line 8: 3, 2, 5, 4, 6, 8, 15, and 9	Line 8: 3, 2, 5, 4, 6, 15, and 9
Line 9: 1, 2, 4, 12, 11, 13, 14, and 10	Line 9: 12, 4, 6, 8, 15, 7, 10, and 14	Line 9: 9, 15, 8, 6, 4, 12, 11, and 10
	Line 10: 1, 2, 4, 12, 11, 13, 10, and 7	Line 10: 12, 4, 6, 8, 15, 7, 10, and 14
		Line 11: 1, 2, 4, 12, 11, 13, 10, and 7
12 lines	**13 lines**	**14 lines**
Line 1: 1, 2, 3, 6, 8, 10, 11, and 13	Line 1: 1, 2, 3, 6, 8, 10, 11, and 13	Line 1: 1, 2, 3, 6, 8, 10, 11, and 13
Line 2: 1, 2, 3, 6, 15, 7, 10, and 8	Line 2: 1, 2, 3, 6, 15, 7, 10, and 8	Line 2: 1, 2, 3, 6, 15, 7, 10, and 8
Line 3: 1, 2, 3, 6, 15, and 9	Line 3: 1, 2, 3, 6, 15, and 9	Line 3: 1, 2, 3, 6, 15, and 9
Line 4: 1, 2, 3, 6, 8, 10, 14, and 13	Line 4: 1, 2, 3, 6, 8, 10, 14, and 13	Line 4: 1, 2, 3, 6, 8, 10, 14, and 13
Line 5: 2, 5, 4, 6, 8, 10, 11, and 13	Line 5: 1, 2, 5, 4, 6, 8, 10, and 11	Line 5: 1, 2, 5, 4, 6, 8, 10, and 11
Line 6: 1, 2, 5, 4, 6, 15, 7, and 10	Line 6: 1, 2, 5, 4, 6, 15, 7, and 10	Line 6: 1, 2, 5, 4, 6, 15, 7, and 10
Line 7: 5, 4, 12, 11, 10, 7, 15, and 9	Line 7: 5, 4, 12, 11, 10, 7, 15, and 9	Line 7: 5, 4, 12, 11, 10, 7, 15, and 9
Line 8: 3, 2, 5, 4, 6, 15, and 9	Line 8: 3, 2, 5, 4, 6, 15, and 9	Line 8: 3, 2, 5, 4, 6, 15, and 9
Line 9: 9, 15, 8, 6, 4, 12, 11, and 10	Line 9: 9, 15, 8, 6, 4, 12, 11, and 10	Line 9: 7, 10, 11, 12, 4, 6, 15, and 9
Line 10: 12, 4, 6, 15, 7, 10, 14, and 13	Line 10: 2, 5, 4, 6, 8, 10, 14, and 13	Line 10: 2, 5, 4, 6, 8, 10, 14, and 13
Line 11: 1, 2, 4, 12, 11, 13, 14, and 10	Line 11: 1, 2, 4, 12, 11, 13, 14, and 10	Line 11: 1, 2, 4, 12, 11, 13, 10, and 7
Line 12: 7, 15, 8, 6, 3, 2, 4, and 12	Line 12: 7, 15, 8, 6, 3, 2, 4, and 12	Line 12: 9, 15, 8, 6, 3, 2, 4, and 12
	Line 13: 12, 4, 6, 15, 7, 10, 14, and 13	Line 13: 7, 15, 6, 4, 12, 11, 13, and 14
		Line 14: 12, 4, 6, 8, 15, 7, 10, and 14

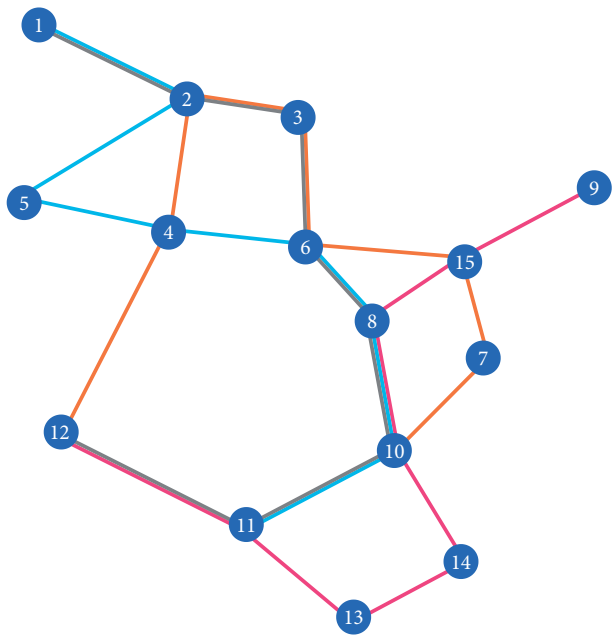

Figure 6: Optimal solution on Mandl's Swiss network with 4 lines with at most 8 nodes each.

result in faster pruning in the branch and bound algorithm. When lines get longer or when there are simply more lines, lower and upper bounds will be updated more often. This results in even more pruning and is an important contributor to keeping the CPU times under control for the higher amount of lines. Another added benefit of having more lines is that the likelihood of not selecting any of the lines at a branch decreases, which results in having to travel less deep to get to a solution. For example, when the absolute lower bound is achievable, this means that every passenger can travel along its shortest path without transfers. Therefore, no matter which OD pair gets chosen to branch upon, there will always be a line selected from the pool. For the instances with eight nodes per line, this happens at fourteen lines. This is thus the minimum amount of lines necessary to reach the lower bound with at most eight nodes per line.

Table 3 shows the optimal solutions for the instances without any restriction on the number of nodes per line, which results in a maximum of fourteen nodes per line in this network. The analysis is the same as for the previous results. Because of the longer lines, the quality of the solution is obviously better than with shorter lines. With thirteen lines, every passenger has a direct connection along its shortest possible path. The actual details of the optimal solutions can be found in Table 4. To the best of our knowledge, only the result for three lines were previously known and published in Fiss and Ritt [66]. Their Mixed Integer Programming implementation required 78992 seconds (on their system) to find the optimal solution with three lines, and they calculated it would take them 77 days to find the optimal solution for four lines. In comparison, our algorithm only needs 15 seconds (for three lines) and 168 seconds (for four lines) to find the optimal solution.

Tables 5 and 6 contain the comparison with the results from literature for a limit of eight and fourteen nodes per line, respectively. In Table 5, the results from our exact algorithm are compared with the results available from [4, 7, 14, 30]. To the best of our knowledge, these papers have the best ATT for this same problem in literature. In Table 6, the results obtained by [8, 10] are chosen since they solve instances with a limit of fourteen nodes per line. These papers report results for instances with four, six, seven, and eight lines. Obviously, the CPU times for the metaheuristics in Table 6 are much lower than for our exact algorithm. Unfortunately, no CPU times are reported for the two approaches reported in Table 6.

Only for four lines with a maximum of eight nodes per line does one of the algorithms from literature find the optimal solution (without knowing it is optimal), but in general, they find a solution close to the optimal. However, the question can be raised whether it is useful to keep competing for finding better metaheuristic solutions on this small network. On this small network, many complex and computationally expensive methods (such as our exact algorithm) can be used to find very good solutions. However, this does not mean the proposed method would work on a realistic network. Hence, researchers should be careful when basing their conclusions only on results found for Mandl's Swiss Network.

5.3. Sorting. In this and following sections, the importance of different parts of the algorithm is evaluated. In Section 4.5, three different sorting methods are introduced. All previous experiments were executed with the main sorting method. In this section, it is analyzed what the impact is of sorting. The main sorting method is compared with the complex sort and the random sort. The sorting itself takes much less than a second (sorting 120 OD pairs based on a criteria). Based on the results from the previous section, only instances with four, eight, and twelve lines are considered in order to somewhat limit the required total CPU time. With this subset, both a very small number of lines as well as a large number of lines are being tested. Eight lines is also chosen because it took a very long time to find the optimal solution in the earlier experiments. The random sort is executed three times, and the best results are reported here. The CPU time per instance (and per execution) is limited to 24 hours. The results are presented in Table 7.

A first obvious conclusion is that the main sort used in the algorithm performs much better than random sort. These results make it clear that sorting your OD pairs is very important in our branch and bound algorithm. The complex sort performs very good when considering a small amount of lines. With four lines, it is significantly faster than the main variant for both line lengths. For eight lines with at most eight nodes, it is a lot faster than the main sort. While with unlimited line lengths, the results are almost equal. For twelve lines, the complex sort can never finish within a day. Hence, it needs a lot more time than the main sort for high line numbers. The main sort is chosen in our algorithm because it can find the optimal solution for all instances.

Table 3: Optimal solutions for Mandl's Swiss network with no limitations to route length.

Number of lines	Max nodes per line	TTT (min)	ATT (min)	CPU time (s)	Solutions checked (M)
3		163430	10.50	15	2
4		159990	10.28	168	44
5		158220	10.16	1641	433
6		157040	10.09	5357	1429
7		156550	10.05	33898	8002
8	14	156090	10.03	20359	5013
9		155930	10.01	14376	1829
10		155840	10.01	8002	816
11		155820	10.01	17590	1025
12		155800	10.01	46067	4036
13		155790	10.01	849	206

Table 4: Optimal solutions for 3 to 13 lines with at most 14 nodes per line.

3 lines	4 lines	5 lines
Line 0: 1, 2, 3, 6, 8, 10, 14, 13, 11, 12, 4, and 5	Line 0: 1, 2, 3, 6, 8, 10, 11, 13, and 14	Line 0: 1, 2, 3, 6, 8, 10, 11, 13, and 14
Line 1: 1, 2, 5, 4, 6, 8, 10, 7, 15, and 9	Line 1: 1, 2, 4, 6, 8, 15, 7, 10, 11, and 12	Line 1: 1, 2, 4, 6, 8, 15, 7, 10, 14, 13, 11, and 12
Line 2: 5, 4, 2, 3, 6, 15, 7, 10, 11, and 12	Line 2: 13, 11, 12, 4, 5, 2, 3, 6, 15, 7, 10, and 14	Line 2: 13, 11, 12, 4, 5, 2, 3, 6, 15, 7, 10, and 14
	Line 3: 1, 2, 5, 4, 6, 8, 10, 7, 15, and 9	Line 3: 1, 2, 5, 4, 6, 8, 10, 7, 15, and 9
		Line 4: 7, 10, 11, 12, 4, 2, 3, 6, 15, and 9
6 lines	**7 lines**	**8 lines**
Line 0: 1, 2, 3, 6, 8, 10, 11, 13, and 14	Line 0: 1, 2, 3, 6, 8, 10, 11, 13, and 14	Line 0: 1, 2, 3, 6, 8, 10, 11, 13, and 14
Line 1: 1, 2, 3, 6, 15, 7, 10, 14, 13, 11, 12, 4, and 5	Line 1: 1, 2, 3, 6, 15, 7, 10, 14, 13, 11, 12, 4, and 5	Line 1: 1, 2, 3, 6, 15, 7, 10, 14, 13, 11, 12, 4, and 5
Line 2: 1, 2, 5, 4, 6, 8, 10, 11, 13, and 14	Line 2: 1, 2, 5, 4, 6, 8, 10, 11, 13, and 14	Line 2: 1, 2, 3, 6, 15, and 9
Line 3: 3, 2, 5, 4, 6, 8, 15, 7, 10, 14, 13, 11, and 12	Line 3: 3, 2, 5, 4, 6, 8, 15, 7, 10, 14, 13, 11, and 12	Line 3: 1, 2, 5, 4, 6, 8, 10, 11, 13, and 14
Line 4: 1, 2, 4, 12, 11, 10, 7, 15, and 9	Line 4: 1, 2, 4, 12, 11, 10, 7, 15, and 9	Line 4: 3, 2, 5, 4, 6, 15, 7, 10, 14, 13, 11, and 12
Line 5: 8, 10, 14, 13, 11, 12, 4, 2, 3, 6, 15, and 9	Line 5: 8, 10, 14, 13, 11, 12, 4, 2, 3, 6, 15, and 9	Line 5: 1, 2, 4, 12, 11, 10, 7, 15, and 9
	Line 6: 7, 10, 11, 12, 4, 6, 8, 15, and 9	Line 6: 7, 10, 11, 12, 4, 6, 8, 15, and 9
		Line 7: 6, 3, 2, 4, 12, 11, 13, 14, 10, 8, 15, and 7
9 lines	**10 lines**	**11 lines**
Line 0: 1, 2, 3, 6, 8, 10, 11, 13, and 14	Line 0: 1, 2, 3, 6, 8, 10, 11, 13, and 14	Line 0: 1, 2, 3, 6, 8, 10, 11, 13, and 14
Line 1: 1, 2, 3, 6, 15, 7, 10, 11, 12, 4, and 5	Line 1: 1, 2, 3, 6, 15, 7, 10, 11, 12, 4, and 5	Line 1: 1, 2, 3, 6, 15, 7, 10, and 8
Line 2: 1, 2, 3, 6, 15, and 9	Line 2: 1, 2, 3, 6, 15, and 9	Line 2: 1, 2, 3, 6, 15, and 9
Line 3: 11, 12, 4, 2, 3, 6, 8, 10, 14, and 13	Line 3: 11, 12, 4, 2, 3, 6, 8, 10, 14, and 13	Line 3: 1, 2, 3, 6, 8, 10, 14, 13, 11, 12, 4, and 5
Line 4: 1, 2, 5, 4, 6, 8, 10, 11, 13, and 14	Line 4: 1, 2, 5, 4, 6, 8, 10, 11, 13, and 14	Line 4: 1, 2, 5, 4, 6, 8, 10, 11, 13, and 14
Line 5: 1, 2, 5, 4, 6, 15, 7, 10, 14, 13, 11, and 12	Line 5: 1, 2, 5, 4, 6, 15, 7, 10, 14, 13, 11, and 12	Line 5: 1, 2, 5, 4, 6, 15, 7, 10, and 8
Line 6: 1, 2, 4, 12, 11, 10, 7, 15, and 9	Line 6: 1, 2, 4, 12, 11, 10, 7, 15, and 9	Line 6: 1, 2, 4, 12, 11, 10, 7, 15, and 9
Line 7: 3, 2, 5, 4, 6, 8, 15, and 9	Line 7: 3, 2, 5, 4, 6, 15, and 9	Line 7: 3, 2, 5, 4, 6, 15, and 9
Line 8: 7, 15, 8, 6, 4, 12, 11, 10, 13, and 14	Line 8: 7, 10, 11, 12, 4, 6, 8, 15, and 9	Line 8: 7, 10, 11, 12, 4, 6, 8, 15, and 9
	Line 9: 7, 15, 8, 6, 4, 12, 11, 10, 13, and 14	Line 9: 11, 12, 4, 6, 15, 7, 10, 14, and 13
		Line 10: 3, 2, 4, 12, 11, 10, 7, 15, 8, and 6
12 lines	**13 lines**	
Line 0: 1, 2, 3, 6, 8, 10, 11, 13, and 14	Line 0: 1, 2, 3, 6, 8, 10, 11, 13, and 14	
Line 1: 1, 2, 3, 6, 15, 7, 10, and 8	Line 1: 1, 2, 3, 6, 15, 7, 10, and 8	
Line 2: 1, 2, 3, 6, 15, and 9	Line 2: 1, 2, 3, 6, 15, and 9	
Line 3: 1, 2, 3, 6, 8, 10, 14, 13, 11, 12, 4, and 5	Line 3: 1, 2, 3, 6, 8, 10, 14, 13, 11, 12, 4, and 5	
Line 4: 1, 2, 5, 4, 6, 8, 10, 11, and 12	Line 4: 1, 2, 5, 4, 6, 8, 10, 11, and 12	
Line 5: 1, 2, 5, 4, 6, 15, 7, 10, and 8	Line 5: 1, 2, 5, 4, 6, 15, 7, 10, and 8	
Line 6: 1, 2, 4, 12, 11, 10, 7, 15, and 9	Line 6: 1, 2, 4, 12, 11, 10, 7, 15, and 9	
Line 7: 1, 2, 5, 4, 6, 15, and 9	Line 7: 1, 2, 5, 4, 6, 15, and 9	
Line 8: 7, 10, 11, 12, 4, 6, 8, 15, and 9	Line 8: 7, 10, 11, 12, 4, 6, 15, and 9	
Line 9: 3, 2, 5, 4, 6, 8, 10, 14, 13, 11, and 12	Line 9: 3, 2, 5, 4, 6, 8, 10, 14, 13, 11, and 12	
Line 10: 3, 2, 4, 12, 11, 10, 7, 15, 8, and 6	Line 10: 3, 2, 4, 12, 11, 10, 7, 15, 8, and 6	
Line 11: 11, 12, 4, 6, 15, 7, 10, 14, and 13	Line 11: 7, 15, 6, 4, 12, 11, 10, and 8	
	Line 12: 7, 10, 14, 13, 11, 12, 4, 6, 8, 15, and 9	

Table 5: Comparison of the exact algorithm with literature for at most 8 nodes per line.

Number of lines	Max nodes per line	Paper	ATT (min)	CPU time (s)
4	8	Exact	10.48	297
		Ahmed	10.48	450
		Islam	10.51	7
		Mandl	12.90	NA
		Chakroborty	11.90	NA
6		Exact	10.18	18856
		Ahmed	10.18	450
		Islam	10.18	7
		Chakroborty	10.48	NA
7		Exact	10.10	39753
		Ahmed	10.10	450
		Islam	10.12	8
		Chakroborty	10.42	NA
8		Exact	10.07	161902
		Ahmed	10.08	450
		Islam	10.07	8
		Chakroborty	10.36	NA

Table 6: Comparison of the exact algorithm with literature for unlimited line length.

Number of lines	Max nodes per line	Paper	ATT (min)	CPU time (s)
4	14	Exact	10.28	168
		Nayeem	10.33	NA
		Wu	10.35	NA
6		Exact	10.09	5357
		Nayeem	10.10	NA
		Wu	10.10	NA
7		Exact	10.05	33898
		Nayeem	10.07	NA
		Wu	10.07	NA
8		Exact	10.03	20359
		Nayeem	—	NA
		Wu	10.04	NA

It can be explained why the complex sort is a lot faster for instances with fewer lines, but it is unable to find an optimal solution for larger number of lines. The complex sort selects OD pairs with a low number of branches but a high demand. This results in slightly more branches than with the main sort, but the average quality of each branch is higher. The high demand means that when a detour from the shortest path is selected, the lower bounds will increase more. This results in more pruning. However, as the amount of lines rises, the number of branches becomes more important for the total CPU time. Remember, from Section 5.2, that it is more likely a line will be selected at each branching point when more lines can be selected in the solution. So, for the early lines, more branches will just result in exponentially more branches in the lower depths. At the same time, more and more (almost all) of the high OD streams will be covered by the actual shortest path. This means there is almost no benefit to selecting these lines first since there will be no extra impact on the lower bound. Hence, it is logical that the complex sort performs better for smaller number of lines.

5.4. Essential Links. In this experiment, the algorithm is also executed without using the concept of essential links. Table 8 shows the results for instances with four, eight, and twelve lines. It is immediately clear that essential links result in a lot less solutions that need to be evaluated. It is expected that this impact would be larger for fewer lines and fewer nodes per line. When there are fewer lines (and/or nodes) in a line plan, then there are fewer edges being served in that line plan. Therefore, it is more likely that certain essential links will not be present in solutions that are evaluated. For up to eight nodes per line, the experiments confirm this trend. For the unlimited line length, this trend is less pronounced. For eight lines, there is only a minimal impact, while for twelve lines, the effect is much larger. In all cases, the essential links do provide a clear positive effect on the CPU time, and for

Table 7: Impact of different sorting methods on the exact algorithm.

Number of lines	Max nodes per line	Sorting method	CPU time (s)	Solutions checked (M)
4	8	Main	297	15
		Complex	37	5
		Random	1666	57
8		Main	161902	25704
		Complex	6812	983
		Random	>86400	>9518
12		Main	16320	1044
		Complex	>86400	>524
		Random	>86400	>11685
4	14	Main	168	44
		Complex	75	25
		Random	3822	701
8		Main	20359	5013
		Complex	21992	4164
		Random	>86400	>18
12		Main	46067	4036
		Complex	>86400	>10418
		Random	>86400	>4867

low line numbers, the CPU time is reduced by 35–45%. We could not measure a significant difference between the precalculation times with or without essential links.

5.5. Direct Link Network. In this section, the algorithm is run with the Change and Go (CNG) network representation instead of the proposed Direct Link Network (DLN) representation. Both algorithms still use the Floyd–Warshall algorithm to evaluate a solution. Table 9 shows the results for four, eight, and twelve lines. All tests ran for up to 24 hours. The number of solutions checked is of course identical. Both methods use the exact same sorting, so the same solutions have to be evaluated. The impact of the DLN is significant and appears to increase when the number of lines or nodes per line increases. This behavior is expected, even for four lines with eight nodes per line, the CPU time increases with a factor four, and for fifteen nodes per line, almost with a factor twenty. The CNG creates an extra node for each stop on each bus line, so more bus lines result in more nodes in the CNG, while the DLN adds new links and only keeps the best links. Therefore, the impact of more and larger lines is more limited. Note that, in literature, Dijkstra's algorithm is used more often than Floyd–Warshall, but we choose to compare with our Floyd–Warshall algorithm to keep the comparison as fair as possible. Nevertheless, some preliminary tests comparing CNG with DLN using Dijkstra's algorithm gave the same type of results as shown here. Obviously, this does not mean that DLN will always perform better than CNG, but these results definitely warrant further research into using the DLN in line planning. It can be concluded that the DLN representation is a major contributor to the speed of our algorithm.

Table 8: The impact of essential links on the exact algorithm.

Number of lines	Max nodes per line	Essential links	CPU time (s)	Solutions checked (M)
4	8	Yes	297	15
		No	545	188
8		Yes	161902	25704
		No	225915	50765
12		Yes	16320	1044
		No	19067	2238
4	14	Yes	168	44
		No	256	106
8		Yes	20359	5013
		No	22975	6180
12		Yes	46067	4036
		No	60121	7206

5.6. Brute Force. We also developed a brute force algorithm to calculate the optimal solution. The brute force algorithm simply tries all combinations of nondominated lines and keeps the best combination. The algorithm started with three lines with eight nodes per line and kept increasing its number of lines until it needs more than 24 h to reach the optimal solution. The results are shown in Table 10. For only three lines, the CPU time is similar to that of our branch and bound algorithm. But, for four lines, it already needs more than ten times as much time. Since the brute force algorithms require exponentially more time when adding an extra line, the algorithm was terminated after it ran for 24 hours for five lines. Thus, finding the optimal solution for five lines or more with this brute force algorithm is not possible in a reasonable amount of time. We conclude that the branch and bound is effective in limiting the amounts of solutions that have to be evaluated.

Table 9: Impact of the Direct Link Network instead of the Change and Go network representation on the exact algorithm.

Number of lines	Max nodes per line	Network model	CPU time (s)	Solutions checked (M)
4	8	DLN	297	15
		CNG	1377	15
8		DLN	161902	25704
		CNG	>86400	>125
12		DLN	16320	1044
		CNG	>86400	>69
4	14	DLN	168	44
		CNG	3277	44
8		DLN	20359	5013
		CNG	>86400	>165
12		DLN	46067	4036
		CNG	>86400	>29

Table 10: Comparison of the branch and bound with a brute force algorithm.

Number of lines	Max nodes per line	Method	CPU time (s)	Solutions checked (M)
3	8	Branch and bound	13	<1
		Brute force	27	16
4		Branch and bound	297	15
		Brute force	3213	1957
5		Branch and bound	3317	466
		Brute force	>86400	>50605

6. Conclusion

In this paper, a novel branch and bound algorithm is developed to find optimal solutions for the uncapacitated line planning problem with integrated passenger routing. The objective is to minimize the total travel time of all passengers, and the available resources are constrained by the number of lines that can be operated and the maximum length of those lines. The algorithm is applied to Mandl's Swiss network [14]. This is by far the most used benchmark instance in line planning research. However, until now, no optimal solutions have been determined for instances with more than three lines on this network. Our algorithm obtains optimal solutions for all available instances on this network, i.e., different number of lines and two different line lengths: for infinite line length and for at most eight nodes per line (5.2). Furthermore, the minimum number of lines necessary to reach the lower bound, where every passenger travels along his/her shortest path without transfers, is determined.

The success of the algorithm is due to a number of new concepts and the way the algorithm is constructed. This is illustrated by the experimental results. By defining all extendable lines as dominated, the size of the pool of feasible lines can already be greatly diminished (4.1). The branch and bound algorithm itself chooses an OD pair to branch on. Every feasible line connecting this OD pair directly with a detour of less than one transfer penalty is considered as a branch as well as not selecting a line. When a line is selected, this line is assumed to provide the optimal routing for the OD pair that was branched upon. This creates many bounds for the OD pair and all possible pairs on the route. The order in which the OD pairs are chosen is also very important (5.3). Three different sorting principles are implemented and tested, and the best way of sorting is implemented in our algorithm. By choosing OD pairs with minimal branches but with high demand first, the CPU time is significantly decreased.

The newly-developed Direct Link Network representation also reduces the required CPU time. For our algorithm, the gains were significant compared to the traditionally used Change and Go Network (5.5). These results definitely warrant to consider the use of the DLN in metaheuristics and other networks as well as looking into adjusting the DLN to deal with frequencies. Finally, in order to decide to actually evaluate a solution or not, the concept of essential links is used. A link is essential if it has to be present in the optimal solution. Essential links can be determined beforehand (4.2). This had a positive effect on the CPU time and was even more pronounced for smaller line numbers and line lengths (5.4).

Finally, by calculating all optimal solutions for the uncapacitated line planning problem, we hope to show that it is no longer useful to play the "up-the-wall" game [67] of trying to beat the best algorithms on this small benchmark network. Actually, it can be concluded that several state-of-the-art solution approaches for the uncapacitated line planning problem obtain near-optimal solutions for almost all these instances. Therefore, aiming to further improve these solutions for these instances is not very useful anymore, but the focus should shift to making the approaches much faster and, more importantly, to develop new concepts for solving larger and more realistic instances.

According to us, our algorithm can be adapted to search for optimal solutions for other variants of the line planning problem, for example, by including frequencies as well. With

frequencies, the line pool generation method has to be adjusted because the dominance criterion is no longer true. Thus, there will be a larger line pool. The passenger route choice will no longer be a shortest path problem, probably resulting in having more links in the DLN. Both of these things will increase the required CPU time. Another possibility is to apply the method to larger networks. Although several problems occurred during preliminary testing with an exact algorithm, it should be possible to integrate at least parts of this algorithm in an efficient (meta) heuristic. If instances are considered with a limited pool of lines generated beforehand, our algorithm should be able to address larger instances as well. Furthermore, we are convinced that both the essential links and the DLN representation can be useful for developing exact and (meta) heuristic algorithms for this or other line planning problems. For instance, the DLN representation could be tested in a metaheuristic setting. Or the essential links could also be updated "dynamically," i.e., every time the lower bound is increased.

Data Availability

All the data of the experiments and the instances is available at the following location: https://www.mech.kuleuven.be/en/cib/lp/mainpage#section-12.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This research was funded by the Research Foundation Flanders, Project G.0853.16N.

References

[1] A. Ceder, *Public Transit Planning and Operation: Modeling, Practice and Behavior*, CRC Press, Boca Raton, FL, USA, 2nd edition, 2016.

[2] R. M. Lusby, J. Larsen, and S. Bull, "A survey on robustness in railway planning," *European Journal of Operational Research*, vol. 266, no. 1, pp. 1–15, 2018.

[3] V. Guihaire and J.-K. Hao, "Transit network design and scheduling: a global review," *Transportation Research Part A: Policy and Practice*, vol. 42, no. 10, pp. 1251–1273, 2008.

[4] L. Ahmed, C. Mumford, and A. Kheiri, "Solving urban transit route design problem using selection hyper-heuristics," *European Journal of Operational Research*, vol. 274, no. 2, pp. 545–559, 2019.

[5] J. J. Blum and T. V. Mathew, "Intelligent agent optimization of urban bus transit system design," *Journal of Computing in Civil Engineering*, vol. 25, no. 5, pp. 357–369, 2011.

[6] J. Duran-Micco, E. Vermeir, and P. Vansteenwegen, "Considering emissions in the transit network design and frequency setting problem with a heterogeneous fleet," *European Journal of Operational Research*, vol. 282, no. 2, pp. 580–592, 2020.

[7] K. A. Islam, I. M. Moosa, J. Mobin, M. A. Nayeem, and M. S. Rahman, "A heuristic aided stochastic beam search algorithm for solving the transit network design problem," *Swarm and Evolutionary Computation*, vol. 46, pp. 154–170, 2019.

[8] M. A. Nayeem, M. K. Rahman, and M. S. Rahman, "Transit network design by genetic algorithm with elitism," *Transportation Research Part C: Emerging Technologies*, vol. 46, pp. 30–45, 2014.

[9] M. Nikolić and D. Teodorović, "Transit network design by bee colony optimization," *Expert Systems with Applications*, vol. 40, no. 15, pp. 5945–5955, 2013.

[10] R. Wu and S. Wang, "Discrete wolf pack search algorithm based transit network design," in *Proceedings of the 2016 7th IEEE International Conference on Software Engineering and Service Science*, pp. 509–512, Beijing, China, 2016.

[11] M. H. Baaj and H. S. Mahmassani, "An AI-based approach for transit route system planning and design," *Journal of Advanced Transportation*, vol. 25, no. 2, pp. 187–209, 1991.

[12] L. Fan and C. L. Mumford, "A metaheuristic approach to the urban transit routing problem," *Journal of Heuristics*, vol. 16, no. 3, pp. 353–372, 2010.

[13] P. N. Kechagiopoulos and G. N. Beligiannis, "Solving the urban transit routing problem using a particle swarm optimization based algorithm," *Applied Soft Computing*, vol. 21, pp. 654–676, 2014.

[14] C. E. Mandl, *Applied Network Optimization*, Academic Press, London, UK, 1979.

[15] C. Mumford, "Data and results 2013," 2013, http://users.cs.cf.ac.uk/C.L.Mumford/Research%20Topics/UTRP/CEC2013Supp/.

[16] I. M. Cooper, M. P. John, R. Lewis, C. L. Mumford, and A. Olden, "Optimising large scale public transport network design problems using mixed-mode parallel multi-objective evolutionary algorithms," in *Proceedings of the 2014 IEEE Congress on Evolutionary Computation (CEC)*, pp. 2841–2848, IEEE, Beijing, China, 2014.

[17] X. Feng, X. Zhu, X. Qian, Y. Jie, F. Ma, and X. Niu, "A new transit network design study in consideration of transfer time composition," *Transportation Research Part D: Transport and Environment*, vol. 66, pp. 85–94, 2019.

[18] M. P. John, C. L. Mumford, and R. Lewis, "An improved multi-objective algorithm for the urban transit routing problem," in *Evolutionary Computation in Combinatorial Optimisation*, C. Blum and G. Ochoa, Eds., Springer, Berlin, Germany, pp. 49–60, 2014.

[19] F. Kılıç and M. Gök, "A demand based route generation algorithm for public transit network design," *Computers & Operations Research*, vol. 51, pp. 21–29, 2014.

[20] J. Yang and Y. Jiang, "Application of modified NSGA-II to the transit network design problem," *Journal of Advanced Transportation*, vol. 2020, Article ID 3753601, 24 pages, 2020.

[21] B. Yao, P. Hu, X. Lu, J. Gao, and M. Zhang, "Transit network design based on travel time reliability," *Transportation Research Part C: Emerging Technologies*, vol. 43, pp. 233–248, 2014.

[22] C. Iliopoulou, K. Kepaptsoglou, and E. Vlahogianni, "Metaheuristics for the transit route network design problem: a review and comparative analysis," *Public Transport*, vol. 11, no. 3, pp. 487–521, 2019.

[23] A. Schöbel and S. Scholl, "Line planning with minimal traveling time," in *Proceedings of the 5th Workshop on Algorithmic Methods and Models for Optimization of Railways*, Palma de Mallorca, Spain, 2006.

[24] R. Z. Farahani, E. Miandoabchi, W. Y. Szeto, and H. Rashidi, "A review of urban transportation network design problems,"

European Journal of Operational Research, vol. 229, no. 2, pp. 281–302, 2013.

[25] O. J. Ibarra-Rojas, F. Delgado, R. Giesen, and J. C. Muñoz, "Planning, operation, and control of bus transport systems: a literature review," *Transportation Research Part B: Methodological*, vol. 77, pp. 38–75, 2015.

[26] K. Kepaptsoglou and M. Karlaftis, "Transit route network design problem: review," *Journal of Transportation Engineering*, vol. 135, no. 8, pp. 491–505, 2009.

[27] A. Schöbel, "Line planning in public transportation: models and methods," *OR Spectrum*, vol. 34, no. 3, pp. 491–510, 2012.

[28] A. Patz, "Die richtige auswahl von verkehrslinien bei grossen strassenbahnnetzen," *Verkehrstechnik*, vol. 51, 1925.

[29] C. E. Mandl, "Evaluation and optimization of urban public transportation networks," *European Journal of Operational Research*, vol. 5, no. 6, pp. 396–404, 1980.

[30] P. Chakroborty and T. Wivedi, "Optimal route network design for transit systems using genetic algorithms," *Engineering Optimization*, vol. 34, no. 1, pp. 83–100, 2002.

[31] S. B. Pattnaik, S. Mohan, and V. M. Tom, "Urban bus transit route network design using genetic algorithm," *Journal of Transportation Engineering*, vol. 124, no. 4, pp. 368–375, 1998.

[32] F. Zhao and X. Zeng, "Simulated annealing-genetic algorithm for transit network optimization," *Journal of Computing in Civil Engineering*, vol. 20, no. 1, pp. 57–68, 2006.

[33] S. Chai and Q. Liang, "An improved NSGA-II algorithm for transit network design and frequency setting problem," *Journal of Advanced Transportation*, vol. 2020, Article ID 2895320, 20 pages, 2020.

[34] E. Vermeir, J. Durán Micco, and P. Vansteenwegen, "The grid based approach, a fast local evaluation technique for line planning," *4OR*, 2021.

[35] A. Mauttone and M. E. Urquhart, "A multi-objective metaheuristic approach for the transit network design problem," *Public Transport*, vol. 1, no. 4, pp. 253–273, 2009.

[36] M. Goerigk and M. Schmidt, "Line planning with user-optimal route choice," *European Journal of Operational Research*, vol. 259, no. 2, pp. 424–436, 2017.

[37] R. Borndörfer, M. Grötschel, and M. E. Pfetsch, "A column-generation approach to line planning in public transport," *Transportation Science*, vol. 41, no. 1, pp. 123–132, 2007.

[38] R. Borndörfer and M. Karbstein, "A direct connection approach to integrated line planning and passenger routing," *OpenAccess Series in Informatics*, vol. 25, pp. 47–57, 2012.

[39] K. Nachtigall and K. Jerosch, "Simultaneous network line planning and traffic assignment," *OpenAccess Series in Informatics*, vol. 9, 2008.

[40] M. E. Schmidt, "Integrating routing decisions in public transportation problems," *Springer Optimization and Its Applications*, Springer, New York, NY, USA, 2014.

[41] M. T. Claessens, N. M. Van Dijk, and P. J. Zwaneveld, "Cost optimal allocation of rail passenger lines," *European Journal of Operational Research*, vol. 110, no. 3, pp. 474–489, 1998.

[42] M. R. Bussieck, *Optimal Lines in Public Transport*, Technische Universität Braunschweig, Braunschweig, Germany, 1998.

[43] J.-W. Goossens, S. van Hoesel, and L. Kroon, "A branch-and-cut approach for solving railway line-planning problems," *Transportation Science*, vol. 38, no. 3, pp. 379–393, 2004.

[44] M. R. Bussieck, T. Lindner, and M. E. Lübbecke, "A fast algorithm for near cost optimal line plans," *Mathematical Methods of Operations Research*, vol. 59, no. 2, pp. 205–220, 2004.

[45] D. Canca, A. De-Los-Santos, G. Laporte, and J. A. Mesa, "Integrated railway rapid transit network design and line planning problem with maximum profit," *Transportation Research Part E: Logistics and Transportation Review*, vol. 127, pp. 1–30, 2019.

[46] H. Dienst, *Linienplanung im spurgeführten personenverkehr mithilfe eines heuristischen verfahrens*, Ph.D. thesis, Technische Universität Braunschweig, Braunschweig, Germany, 1978.

[47] M. R. Bussieck, U. Zimmerman, and P. Kreuzer, "Optimal lines for railway systems," *European Journal of Operational Research*, vol. 96, pp. 54–63, 1996.

[48] A. Bouma and C. Oltrogge, "Linienplanung und simulation für öffentliche verkehrswege in praxis und theorie," *Eisenbahntechnische Runschau*, vol. 43, pp. 369–378, 1994.

[49] J. F. Guan, H. Yang, and S. C. Wirasinghe, "Simultaneous optimization of transit line configuration and passenger line assignment," *Transportation Research Part B: Methodological*, vol. 40, no. 10, pp. 885–902, 2006.

[50] M. Schmidt and A. Schöbel, "The complexity of integrating passenger routing decisions in public transportation models," *Networks*, vol. 65, no. 3, pp. 228–243, 2015.

[51] E. Cipriani, S. Gori, and M. Petrelli, "Transit network design: a procedure and an application to a large urban area," *Transportation Research Part C: Emerging Technologies*, vol. 20, no. 1, pp. 3–14, 2012.

[52] I. Constantin and M. Florian, "Optimizing frequencies in a transit network: a nonlinear bi-level programming approach," *International Transactions in Operational Research*, vol. 2, no. 2, pp. 149–164, 1995.

[53] C. L. Mumford, "New heuristic and evolutionary operators for the multi-objective urban transit routing problem," in *Proceedings of the 2013 IEEE Congress on Evolutionary Computation*, pp. 939–946, IEEE, Cancun, Mexico, 2013.

[54] B. Beltran, S. Carrese, E. Cipriani, and M. Petrelli, "Transit network design with allocation of green vehicles: a genetic algorithm approach," *Transportation Research Part C: Emerging Technologies*, vol. 17, no. 5, pp. 475–483, 2009.

[55] W. Fan and R. B. Machemehl, "Optimal transit route network design problem with variable transit demand: genetic algorithm approach," *Journal of Transportation Engineering*, vol. 132, no. 1, pp. 40–51, 2006.

[56] H. Shimamoto, N. Murayama, A. Fujiwara, and J. Zhang, "Evaluation of an existing bus network using a transit network optimisation model: a case study of the Hiroshima city bus network," *Transportation*, vol. 37, no. 5, pp. 801–823, 2010.

[57] H. Shimamoto, J.-D. Schmöcker, and F. Kurauchi, "Optimisation of a bus network configuration and frequency considering the common lines problem," *Journal of Transportation Technologies*, vol. 2, no. 3, pp. 220–229, 2012.

[58] R. O. Arbex and C. B. da Cunha, "Efficient transit network design and frequencies setting multi-objective optimization by alternating objective genetic algorithm," *Transportation Research Part B: Methodological*, vol. 81, pp. 355–376, 2015.

[59] A. T. Buba and L. S. Lee, "A differential evolution for simultaneous transit network design and frequency setting problem," *Expert Systems with Applications*, vol. 106, pp. 277–289, 2018.

[60] M. Nikolić and D. Teodorović, "A simultaneous transit network design and frequency setting: computing with bees," *Expert Systems with Applications*, vol. 41, no. 16, pp. 7200–7209, 2014.

[61] M. Owais, M. K. Osman, and G. Moussa, "Multi-objective transit route network design as set covering problem," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 3, pp. 670–679, 2016.

[62] A. Ceder and N. H. M. Wilson, "Bus network design," *Transportation Research Part B: Methodological*, vol. 20, no. 4, pp. 331–344, 1986.

[63] H. Fu, L. Nie, L. Meng, B. R. Sperry, and Z. He, "A hierarchical line planning approach for a large-scale high speed rail network: the China case," *Transportation Research Part A: Policy and Practice*, vol. 75, pp. 61–83, 2015.

[64] D. B. Johnson, "Efficient algorithms for shortest paths in sparse networks," *Journal of the ACM*, vol. 24, no. 1, pp. 1–13, 1977.

[65] R. R. Williams, "Faster all-pairs shortest paths via circuit complexity," *SIAM Journal on Computing*, vol. 47, no. 5, pp. 1965–1985, 2018.

[66] B. C. Fiss and M. Ritt, "Exact and metaheuristic algorithms for the urban transit routing problem," in *Proceedings of the Congreso Latino-Iberoamericano de Investigacion Operativa, Simposio Brasileiro de Pesquisa Operacional*, pp. 3717–3728, Rio de Janeiro, Brazil, 2012.

[67] K. Sörensen, "Metaheuristics-the metaphor exposed," *International Transactions in Operational Research*, vol. 22, no. 1, pp. 3–18, 2015.