Università di Pisa

Dipartimento di Informatica

Technical Report

A Matheuristic for Integrated Timetabling and Vehicle Scheduling

Samuela Carosi Antonio Frangioni Laura Galli Leopoldo Girardi
Giuliano Vallese

A Matheuristic for Integrated Timetabling and Vehicle Scheduling

Samuela Carosi [*] Antonio Frangioni [†] Laura Galli [†‡] Leopoldo Girardi [*] Giuliano Vallese [*]

Abstract

Planning a public transportation system is a complex process, which is usually broken down in several phases, performed in sequence. Most often, the trips required to cover a service with the desired frequency (headway) are decided early on, while the vehicles needed to cover these trips are determined at a later stage. This potentially leads to requiring a larger number of vehicles (and, therefore, drivers) that would be possible if the two decisions were performed simultaneously. We propose a multicommodity-flow type model for integrated timetabling and vehicle scheduling. Since the model is large-scale and cannot be solved by off-the-shelf tools with the efficiency required by planners, we propose a diving-type matheuristic approach for the problem. We report on the efficiency and effectiveness of two variants of the proposed approach, differing on how the continuous relaxation of the problem is solved, to tackle real-world instances of bus transport planning problem originating from customers of *M.A.I.O.R.*, a leading company providing services and advanced decision-support systems to public transport authorities and operators. The results show that the approach can be used to aid even experienced planners in either obtaining better solutions, or obtaining them faster and with less effort, or both.

Keywords*: Public transport, timetabling, vehicle-scheduling, integrated approach, matheuristic*

1 Introduction

Public transportation companies often face complex logistic problems. In particular, vehicles and crews represent expensive resources for the operators, that require efficient utilization. Planning in a public transportation system is usually decomposed into stages, that are solved in sequence, namely: Network Design (ND), TimeTabling (TT), Vehicle Scheduling (VS) and Crew Scheduling (CS). The first two steps define the type of service to be offered: ND determines the set of lines (and connections) and how often the service is offered along the lines, while TT defines the departure and arrival time of the individual trips on the given lines, in order to meet the desired frequency of service. The last two steps are, instead, devoted to resource allocation: VS is the assignment between buses and trips, such that each trip is covered by exactly one bus and the schedules of all the vehicles are feasible, while CS is the assignment of crews and trips, such that each trip is covered by a crew and all the crew schedules satisfy the required logical and legal restrictions. We refer the reader to [11] for a detailed description of the various stages, and to [20] for a global review of the crucial strategic and tactical steps of transit planning.

There is a vast literature addressing each one of the above steps individually. Yet, because of the interdependence of the stages, planning in sequence possibly produces suboptimal solutions. This is in particular true for the vehicles and drivers needed, that are only determined in the later steps of the planning process. Unfortunately, decomposing into stages is often necessary to make the solution time

[*]OR division, M.A.I.O.R. Srl, Lucca, Italy. E-mail: {samuela.carosi, leopoldo.girardi, giuliano.vallese}@maior.it

[†]Dipartimento di Informatica, Università di Pisa, Largo B. Pontecorvo 3, 56127 Pisa, Italy. E-mail: {antonio.frangioni, laura.galli}@unipi.it

[‡]Corresponding author

compatible with the requirements of the planners. It is generally argued [8] that the two intermediate stages, TT and VS, are "the bulk" of the decision process. Indeed, recent developments in transit planning, including this work, focus on the *integration* of these two steps.

In particular, our contribution consists in a new model for the Integrated Timetabling and Vehicle Scheduling (ITTVS) problem. Under some assumptions on the VS constraints, that are particularly reasonable for the urban planning context and can be somewhat relaxed, the model is a compact multicommodity-flow type problem; however, its size and the relative weakness of the continuous relaxation are such that the problem cannot be solved by off-the-shelf tools with the efficiency required by the planners. We therefore also propose a diving-type matheuristic approach for the problem, which produces good-quality solutions in reduced time. We report experiments on several real-world ITTVS instances originating from customers of *M.A.I.O.R.*, a leading company providing services and advanced decision support systems to public transport authorities and operators, showing that good-quality solutions—in particular, if compared with those manually constructed by experts of the transport companies and actually used in operations—can be obtained with a reasonable computational effort. The variant of the approach where the continuous relaxation of the model is tackled by forming its Lagrangian relaxation w.r.t. the linking constraints, and approximately solving the corresponding Lagrangian Dual by means of a bundle-type method, appears to be particularly promising as the size and complexity of the instances grow.

The structure of the paper is as follows. In Section 2, we provide a general description of the TT and VS as individual steps, and we review the literature dealing with attempts at integrating the two; the derived taxonomy allows us to frame our contribution. Then, Section 3 presents the base case scenario for our real-world application, which is mathematically formulated in Section 4. Section 5 discusses some important extensions to the base case scenario. Our *matheuristic* approach is described in Section 6, and computational results are discussed in Section 7. Finally, in Section 8, we draw some conclusions.

2 Literature Review

2.1 Timetabling

Timetabling (TT) is the process of creating a schedule starting from the route network and the desired frequency of service. The result is a set of trips, with the scheduled times at the terminals and major points on the routes, a.k.a. the *timetable*. Timetabling can be *periodic* ("clock-face") or *non-periodic*. If the order of the events is fixed, the latter can be efficiently solved by shortest path techniques. If events appear periodically, an ordering is not possible, this is why the *periodic event scheduling problem* (PESP) is $\mathcal{NP}$-hard [30]. In the case of non-periodic TT one usually measures the *headway* of a line, i.e., the time separating the service at its main stop by consecutive runs; this specifies how often bus service should be offered, and is clearly the inverse of the frequency over a time period, usually considered in "clock-face" timetabling.

The TT problem aims at finding "good quality" timetables, from the viewpoint of users of the transportation service. This may mean different things. Perhaps the simplest one is *regularity*, whereby one seeks to find a timetable where the trips have exactly the frequency/headway required for the line they belong to (in the corresponding time window), or at least the distance of the actual frequency from the desired one is minimized. This is the only reasonable measure if the topology consists of a single (albeit, possibly, "complex") line, as in our experiments. However, when multiple lines are considered, the *transfer coordination* or *synchronization* variant is also studied, where one is rather interested in finding schedules that minimize transfer and/or waiting time of passengers (or other synchronization measures) at the stops connecting different lines. That is, the aim is to coordinate the trips on different lines; clearly, this requires modelling passengers' waiting and transfer activities during vehicle changes.

In the context of transit planning, TT is included within *operational* planning. The reason is twofold; (*i*) timetabling occurs frequently (e.g., every 3-6 months); (*ii*) it is from the timetables that vehicle and crew schedules are constructed. Yet, the goal of timetabling is a *tactical* one, since it aims at *optimizing passengers' service*. This is in contrast to the VS and CS, that are typically intended to *minimize operating costs*.

2.2 Vehicle Scheduling

If the lines and the timetable are given, so is the set of *trips*, i.e., sequences of arrival/departure times at each stop of a given line, that must be operated by the same vehicle. The set of trips is the input of VS, which aims at optimally covering them, typically minimizing the number of vehicles needed and/or some other measure of the required effort, such as deadheads (i.e., vehicle movements that do not constitute transportation service) or other operating costs, while meeting all operational constraints. VS plays an important role in the public transport planning process, since it is the first planning step, where the primary focus is put on *minimizing* costs, while previous steps typically focus on optimizing passenger service. The vehicle scheduling problem is the task of building an optimal set of sequences of trips, each sequence—called *vehicle schedule*—to be performed by an individual vehicle, such that each trip of a given timetable is covered by exactly one sequence. A sequence of trips assigned to a vehicle results in a *vehicle route*, that might very well serve several lines (*interlining*). *Multi-depot* VS is $\mathcal{NP}$-hard (cf. [6]), while the *single-depot* case can be solved in polynomial time [3], provided there are no constraints on how a chain can be formed, apart from compatibility between two trips (taking into account max/min waiting time at terminals and/or deadheading, if allowed). More complex variants also consider different types of vehicles (e.g., number of seats, level of comfort, etc.)

2.3 Literature Review

With only one exception, all the works in the literature considering integrated timetabling and vehicle scheduling in urban public transport deal with the *transfer coordination* version of the TT, i.e., where the objective is to minimize the transfer and waiting time for passengers. To the best of our knowledge, the first two papers are [7] and [9]. The former presents a 4-step sequential approach with a single feedback loop that determines a timetable and the corresponding vehicle schedules. The solution approach of the latter, instead, is based on a genetic algorithm to simultaneously optimize the fleet size without interlining (i.e., each bus can serve only one line) and the waiting and transfer time of passengers.

In general, a crucial characteristic of all approaches is that the integrated problem has a *bi-objective* nature; that is, it aims simultaneously at maximizing the *timetable quality* from the passengers' point of view, and minimizing the *operating cost* of vehicle schedules from the service provider's point of view. Clearly, these two objectives are potentially in contrast to each other; thus, a main decision, when developing an integrated model is on how the interaction between the two contrasting objective functions should be managed. Correspondingly, we subdivide all the articles in the literature according to the strategy they adopt in this respect:

- *Shifting.* An important stream of research is based on the idea of solving the VS problem allowing some flexibility to change the timetable, thus leading to the *Vehicle Scheduling with Time-Windows* (VSP-TW) problem. That is, the timetable is given as an input, and arrival times can only be modified (shifted) by a small amount, in order to allow for cheaper vehicle schedules. Clearly, this approach prioritises the service provider's objective function (operating cost); however, it does so, because the quality of the timetable is somewhat guaranteed by the fact that only minor modifications, w.r.t. the nominal one, are allowed. Hence, in multi-objective parlance, these methods are akin to *budgeting* ones, where one objective is optimized subject to the constraint that the other one cannot become worse than a given threshold (although in this case the threshold is only indirectly specified).

- *Weighting.* The other classical approach in multi-objective optimization consists in having, as objective function, the *weighted* sum of the two original ones. As usual, the issue with this kind of approaches is that of finding weights that accurately represent the preferences of the decision maker.

- *Pareto-front.* To account for the inherent difficulty of the two previous approaches, i.e., that of selecting either an appropriate budget or appropriate weights, it is possible to try to produce a set of Pareto-optimal (i.e., non dominated) solutions. This can be done, for example, by solving the budgeted/weighted versions of the problem with several choices of the budget/weights, or, alternatively, using *population-based algorithms*, as they naturally generate multiple solutions.

- *Bilevel programming.* This approach takes a different stance, where one of the two objective functions is that of the *leader* (say, the service provider), who optimizes it, while the *followers* (say, the users) react by optimizing their own (say, their travel time), subject to the leader choices.
- *Reordering.* Finally, in this specific context, the idea has been proposed that it might be possible to obtain "more integrated" solutions by simply reordering the classic sequence of the planning steps outlined at the beginning of this section.

We will now briefly describe all the papers in the literature as subdivided among the five above categories. It might be appropriate to mention at this point that, due to the complexity of the problem, most of these studies propose *meta-heuristic* algorithms such as Iterated Local Search (ITL), Tabu-Search (TS), Large Neighborhood Search (LNS), Genetic Algorithms (GA), and Simulated Annealling (SA).

2.3.1 Shifting

This kind of approach can be traced back to the seminal paper [23], which considers (small) departure time windows in which the departure time of a service trip can be shifted, and use a discrete *time-space network* to determine feasible trip combinations. The solution approach is based on column generation together with heuristics. The time-space network model of [23] is extended in [31], where a hierarchical approach for VS is developed, combining mathematical programming models, to optimize the type and the number of vehicles for each trip, with a SA approach, that allows the trip starting times to be shifted in time. In the network, vertices are departures and arrivals of a vehicle at a specific time and station (or the depot), such as the beginning or end of a service trip, and edges link two actions that can be performed by the same vehicle. A vehicle schedule corresponds to a flow through the network, so that the computation of the optimal vehicle schedule can be performed by calculating a minimal cost circulation through the network, with additional constraints guaranteeing that all service trips are performed exactly once. Similarly, the use of VSP-TW in the context of tactical timetable analysis is discussed in [5], where it is suggested to model the VSP-TW as a *Vehicle Routing Problem with Time Windows* (VRP-TW) and to estimate the potential of vehicle savings for a given timetable by allowing wider departure time windows (up to 20 minutes) for service trips. Recently, [15] proposes a matheuristic that combines the idea of *shifting* with that of *weighting*. The algorithm iteratively solves a bi-objective mathematical formulation (minimization of passenger transfers and operational costs) of the ITTVS allowing timetable modifications for a subset of timetabled trips, while solving the full VS problem.

2.3.2 Weighting

The ITTVS with *time windows and balanced departure times* is studied in [29]; the problem is modeled as a VRP-TW, that includes the balancing of trips departure times and minimization of deadheads in the objective function, and it is solved by a hybrid LNS approach that decomposes the problem into a scheduling and a balancing component. The *Simultaneous Vehicle Scheduling and Passenger Service Problem* (SVSPSP) has been defined for the first time in [28], where an integrated solution approach is proposed, based on the LNS used in [12] to solve the multiple depot vehicle scheduling problem (MDSVP); the approach is tested on the Greater Copenhagen Area. A solution approach based on TS is presented in [22], where at each iteration the timetable is altered and the optimal trip assignment is recomputed solving a linear quasi-assignment problem. An interactive tool called `NetPlan` is described in [13, 14] that integrates timetabling and vehicle scheduling. The tool is developed by `GIRO`, a privately owned company based in Montreal that provides software and services to plan and manage public transport operations; however, the papers provide little detail about the heuristics used.

2.3.3 Pareto-front

Two integer linear programming models for TT and VS are defined in [2] and combined in a bi-objective integrated model that is solved repeatedly using a *budgeting* approach. A ITTVS model (without interlining) is presented in [32] and solved by the direct application of a *multi-objective* GA.

2.3.4 Bi-level

A bi-level ITTVS integer programming model is developed in [1] and it is solved using a specialized TS algorithmic framework. A more complex bi-objective and bi-level approach is presented in [24, 25] to study how much the changes on timetable and vehicle scheduling affect users trips choice behaviour. In the model, the upper level is a service provider, that creates timetables and vehicle schedules to minimize total operating costs and passenger waiting/travel time, while the lower level are public transport users, who choose their travel paths in a user optimal manner, responding to the operator decision (transit assignment problem).

2.3.5 Reordering

A "reverse shifting" approach is proposed in [21] and tested on real-life instances from France; the input is the current timetable, vehicle and crew schedule, and the timetable is adjusted by a TS approach keeping the vehicle and driver schedules fixed. In [27] the process starts by designing the vehicle routes; then these routes are interpreted as lines and the corresponding frequency is defined, finally the timetabling phase assigns an arrival and a departure time to each stop of the route. The objective function is designed in order to measure the "attractiveness" for passengers, using an origin-destination matrix of potential transport demands and maximizing the probability that a (potential) traveler between two locations decides to use public transportation rather than a private one. The heuristic is applied to a case study that optimizes the local bus system in Gottingen.

2.4 Contributions of this paper

As already mentioned, almost all the previously cited articles focus on the *transfer coordination* version of the TT, save for [29], where *regularity* (i.e., minimizing the deviation from the desired headways) is considered. Also, almost all the contributions use meta-heuristics, save for [15], where a matheuristic approach is developed for an integrated bi-objective formulation (but with the transfer coordination objective function).

The ITTVS problem at *MAIOR* is non-periodic, with regularity objective function for the TT component, and single depot, single vehicle type VS model; the characteristic features of our problem are described in Section 3. The contributions of the paper are the following. First, we consider a real-world bus planning application at *MAIOR* and present an integrated solution approach for two steps (i.e., TT and VS), that were previously solved in sequence by customers of the company. This allows us to test our integrated approach on real-world instances provided by Italian public transport providers, comparing them with those previously produced by the sequential approach. Thus, we are able not only to compare the objective values, showing that the integrated approach significantly reduces them, but also to have our solutions evaluated by expert transport planners, which certified them to be of "better quality", according the their judgement. This is important, because our integrated approach is based on weighting, and the proper selection of weights is crucial for the practical quality of the solution. Moreover, to our knowledge, our approach is the first matheuristic for ITTVS with regularity objective function. Finally, we consider some extensions for dealing with "complex" single-line topologies and constraints on the number of vehicles, which are important for the practical applicability of the approach in some customers' environments.

3 Problem Description

This section describes the specific characteristics of the "base case" ITTVS problem at *MAIOR*, where the topology is that of a *simple single line*. This is only for simplicity of exposition, in that the mathematical formulation presented in Section 4 immediately extends to multiple independent lines (that is, independent from the TT side, while potentially linked in the VS one). Less trivial extensions are shown in Section 5, in particular for when multiple routes (besides the two obvious ones) actually pertain to the same line (i.e., a *complex single line*).

The main input to the integrated TT-VS problem is a *public transportation network* (PTN), a set of potential trips $\mathcal{T}$, and the desired (a.k.a. *ideal*) headways for each of the *time windows* in which the operating interval is subdivided, and for each direction. In general, a PTN is given under the form of a graph, where the nodes correspond to bus stops or depots, and the links correspond to direct bus transits; however, the actual graph description of the PTN is inconsequential for our treatment. A *simple (single) line* is a bi-directional path AB in the PTN between two *terminals* A and B (i.e., start/end stops of a line). Usually a simple line has two directions, called *in-bound* and *out-bound* and denoted by $\mathcal{D} = \{ \vec{AB} , \vec{BA} \}$, respectively; however, a *circular* line (where $A = B$) may have only one direction. In more general cases, a single line may comprise multiple routes or *patterns* for each direction, as discussed in Section 5; however, in this paragraph patterns and directions coincide, as shown in Figure 1. A *trip* corresponds to a pattern/direction in the PTN that has to be operated by some vehicle at a given time. Since each trip belongs to a given pattern/direction, we define $\mathcal{T} = [\, \mathcal{T}_d \,]_{d \in \mathcal{D}}$ as the "direction partition" of $\mathcal{T}$. Each trip $i \in \mathcal{T}$ is characterized by a start and end terminal, denoted by $sn(i)$ and $en(i)$, respectively, while the corresponding departure time from $sn(i)$ and arrival time at $en(i)$, are denoted by $st(i)$ and $et(i)$, respectively. Even in the case of a simple single line, for VS purposes it is necessary to consider in the PTN, besides the terminal nodes A and B, also the *single deposit* node O (but not any other intermediate stop of the line).

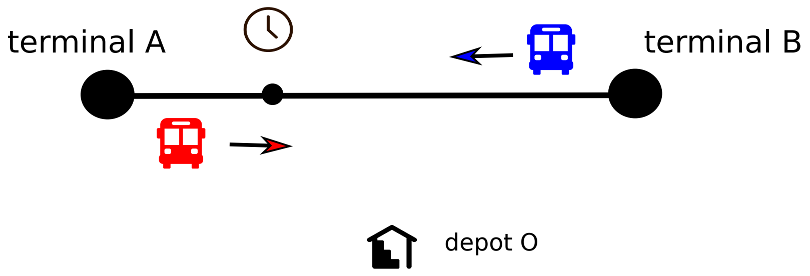

Figure 1: Simple single line

In the following, we will denote by N the set of terminals of the involved lines (say, $N = \{ A , B \}$) and by $N^+ = N \cup \{ O \}$. For each direction, a *main stop* is identified, symbolized by a "clock" in Figure 1, which is used to calculate the *headways.* Although the figure may suggest that the main stop needs be the same for the two directions of a simple line, this is not necessarily true in practice (especially since the stops along the two directions could be disjoint). Also, the main stop may, or may not, coincide with one of the terminals of the line.

TT components. Each trip $i \in \mathcal{T}$ is associated to a uniquely identified pattern/direction $d(i)$. The trip specifies the arrival time at each stop of the pattern, including the arrival time $a(i)$ at the main stop of $d(i)$. Since we assume only one vehicle type, the arrival times of a given trip are the same for all the vehicles. A *timetable* π_d *for a direction* $d \in \mathcal{D}$ is a subset of its potential input trips $\mathcal{T}_d$; a timetable is then just the union of $|\mathcal{D}|$ (independent) timetables, one for each direction, i.e., $\pi = [\, \pi_d \,]_{d \in \mathcal{D}}$. Given a timetable π, the (actual) headways of a direction d w.r.t. π are the times separating each two consecutive trips i, j in π_d passing at its main stop, i.e., $a(j) - a(i)$. In our non-periodic planning, a time horizon T is given (say 5:00–24:00, i.e., each day is treated independently). As the desired frequency of service typically varies along the day, T is partitioned into k *time windows* defined by $k + 1$ time instants t_0, t_1, …, t_k, where t_0 and t_k are the initial and final time instants of T. We will denote by $h(i)$ the time window in which trip i happens; for simplicity we will only deal with the case in which trips are completely contained in a time window, but only minor changes (whose details are not worth reporting) are required to account for "border effects" when they do not. For each time window h and each direction $d \in \mathcal{D}$, we are given the ideal headway I^h_d, together with *minimim and maximum headways* $\underline{I}^h_d \leq I^h_d \leq \bar{I}^h_d$. A *feasible timetable* $\pi_d \subset \mathcal{T}_d$ for a direction $d \in \mathcal{D}$ is a timetable such that:

- the (actual) headway of each two consecutive trips i and j in π_d is feasible, i.e., $a(j) - a(i) \in [\, \underline{I}^{h(i)}_d \, , \, \bar{I}^{h(i)}_d \,]$ (with a minor variation if $h(i) \neq h(j)$);
- the *first* and the *last trip* of π_d belong to given subsets $\mathcal{T}^{ini}_d$ and $\mathcal{T}^{fin}_d$ of *initial* and *final* trips, specified as an input to the problem.

To evaluate the quality of a timetable, a *penalty function* is given specifying how to compute the cost of the deviation of a feasible actual headway $\bar{a} \in [\underline{I}_d^h, \bar{I}_d^h]$ from the ideal one I_d^h. The actual form of this function is immaterial for our model, just assuming the trivial properties that the penalty is zero if $\bar{a} = I_d^h$, and larger than zero (typically, nondecreasing in $|\bar{a} - I_d^h|$) otherwise.

VS components. In the VS literature, traveling between two trips without passengers on board is called a *deadhead trip*. In particular, a vehicle leaving a depot to reach the start-terminal of a trip is said to be performing a *pull-out trip*; similarly, it performs a *pull-in trip* when it returns to the depot from the end-terminal of a trip. For each node $n \in N^+$ and for each time window h we are given a *minimum* and a *maximum stopping-time*, denoted by $\delta_{min,n}^h$ and $\delta_{max,n}^h$, respectively; however, we typically assume that there is no maximum stopping time at the depot, i.e., $\delta_{max,O}^h = +\infty$ for all h. For each terminal $n \in N$ and for each time window h, we are also given the *travel time* for a pull-in and pull-out trip, denoted by t_{n+}^h and t_{n-}^h, respectively. In general, two trips are said to be *compatible* if they can be covered consecutively by the same vehicle. In our application, we distinguish two types of compatibilities between two trips $i, j \in \mathcal{T}$:

- *in-line compatibility* means that $en(i) = sn(j)$, i.e., trip j starts at the same terminal in which i ends, and $\delta_{min,en(i)}^{h(i)} \leq st(j) - et(i) \leq \delta_{max,en(i)}^{h(i)}$, i.e., the waiting time at the terminal between trip the end of i and the start of trip j is feasible;
- *out-line compatibility* means that $en(i) \neq sn(j)$ and $st(j) - et(i) \geq t_{en(i)+}^{h(i)} + \delta_{min,O}^{h(i)} + t_{sn(j)-}^{h(j)}$; in other words, there must be enough time between the end of trip i and the start of trip j to perform a pull-in trip from $en(i)$, wait the minimum amount of time at the depot, and then perform a pull-out trip towards $sn(j)$. Note that pull-in and pull-out trips are not included in $\mathcal{T}$, as they are not (passenger) service trips (i.e., no passengers on board).

In our case study deadheading is not allowed, so that if $en(i) \neq sn(j)$ the vehicle cannot move directly from one terminal to the other, but it must perform an out-line compatibility. Yet, deadhead trips could make sense and therefore be allowed, subject to time compatibility, without this impacting the general structure of our model (barring some specific details discussed later on). A *feasible schedule for a vehicle* is composed of an initial pull-out trip, a sequence of compatible (service) trips in $\mathcal{T}$, possibly separated by pull-in/out trips, and a final pull-in trip to return to the depot. In general, feasible schedules for a vehicle can be seen as sequences of *vehicle blocks*, where each block consists of a sequence of (service) trips in $\mathcal{T}$, that starts and ends at the depot without returning to it in the middle of the sequence; if deadheading is allowed, deadhead trips may need to be specified to complete the description of a feasible schedule for a vehicle. A feasible *vehicle schedule* Ω is a subset of the input potential trips $\mathcal{T}$ that can be partitioned in feasible schedules for single vehicles, possibly satisfying a constraint on the maximum number of vehicles to be used if it is imposed.

The objective of our integrated problem is to provide a solution that optimally balances the service provider cost (VS objective) and the users satisfaction (TT objective). The latter is simply captured by minimizing the sum of the costs of all the deviations between the actual headways and the desired ones, each one measured by the penalty function alluded to above. The former is somewhat more complex. Since one of the main costs for the service provider is usually due to the number of vehicles used, the primary VS objective is the minimization of the number of bus schedules. Two secondary measures of the service provider cost are the time spent by the vehicles waiting at the terminals (for drivers will typically have to man them even when stationary, thus increasing labour cost), and the time spent by the vehicles performing pull-in and pull-out—and deadhead, if allowed—trips (for the same reason as above, plus the fact that vehicles typically consume some fuel). Note that, if dead-heading is not allowed, the latter secondary objective typically minimizes the number of vehicle blocks. The relative importance of these terms is defined by weighting parameters in the VS objective function; this is in addition to the weights given to the two different overall objective functions (TT and VS), but the selection of the sub-weights for the VS part is typically done even when solving the problem by separate phases, and therefore these are well-established in practice (also because they can ultimately be reduced to monetary costs). The selection of the weights for the primary objective functions is more delicate, which is why judgement by experts was required to evaluate the solutions produced by the integrated approach before the results could be deemed satisfactory.

4 Mathematical Model

We now present the mathematical model for the ITTVS as described in the previous section. We will often make reference to the "base case" scenario of a simple single line for illustration, although the model readily extends to any number of lines (patterns). The model consists of $|\mathcal{D}|$ *TT sub-problems*, one for each direction, a single *VS sub-problem*, and some *linking* constraints, that guarantee integration. The TT sub-problems select an optimal subset of trips $\mathcal{T}^* \subset \mathcal{T}$, corresponding to feasible timetables $\pi^* = [\, \pi_d^* \,]_{d \in \mathcal{D}}$ for all directions, that minimize the total cost of deviation from the ideal headways. The VS sub-problem constructs a feasible vehicle schedule Ω^* with minimum operator cost out of the selected trips $\mathcal{T}^*$; in other words, Ω^* is a *vehicle schedule cover* of π^*. Clearly, the subproblems are not independent since the vehicle schedule depends on the choice of $\mathcal{T}^*$, which is what the linking constraints provide by ensuring that the trips covered by Ω^* correspond to all and only the trips used in the timetable.

We propose a *Mixed Integer Linear Programming (MILP)* multicommodity flow-type model, based on node-arc formulations where arc flow variables represent either the timetables or the vehicles schedule. That is, we construct one directed graph for each of the TT subproblems (direction d) and one directed graph for the VS subproblem, as described in Subsection 4.1 and Subsection 4.2, respectively. Finally, the integrated MILP formulation, comprising the linking constraints, is shown in Subsection 4.3.

4.1 TT model

The TT model is based on representing feasible timetables in terms of paths on a directed *TT graph* $G_d^{TT} = (\, N_d^{TT} \,,\, A_d^{TT} \,)$, which is a *compatibility graph.* For a given direction $d \in \mathcal{D}$, the nodes of the corresponding TT graph represent the trips in $\mathcal{T}_d$ plus a dummy source O_d^- and a dummy sink O_d^+: $N_d^{TT} = \mathcal{T}_d \cup \{\, O_d^- \,,\, O_d^+ \,\}$. The arcs in A_d^{TT} leaving the source node O_d^- end in the nodes corresponding to the set of potential initial trips $\mathcal{T}_d^{ini}$, and symmetrically for those entering the sink node O_d^+. An arc $(i,j) \in A_d^{TT}$ between two trips $i, j \in \mathcal{T}_d$ exists if and only if i and j are neither "too close" or "too far apart", i.e., the corresponding headway is feasible. Its cost is computed off-line with the selected penalty function, which can therefore be arbitrary (in practice, a piecewise-linear function with a fixed cost is used). It is trivial to see that G_d^{TT} is acyclic and that a path between O_d^- and O_d^+ in G_d^{TT} corresponds to a feasible timetable π_d, the cost of the path (sum of the costs of the arcs) being the total cost of violation of ideal headways. Being G_d^{TT} acyclic, each TT sub-problem—were they independent, which they are not—could be easily solved as an *acyclic shortest path (SP)* problem, whose complexity is linear in $|A_d^{TT}|$ and therefore at worst quadratic in $|\mathcal{T}_d|$ (but in practice basically linear in $|\mathcal{T}_d|$, since many trips are not compatible due to the constraints on the minimum and maximum headway).

Example 1. Consider the small example in Figure 2 with 5 trips. G_d^{TT} consists of two dummy nodes (*source* O_d^- and *sink* O_d^+) and 5 trip nodes, for simplicity all belonging to the same time window. The time indicated inside the trip nodes represents the arrival time $a(i)$ at the *main stop.* The ideal headway is 2 minutes, with minimum and maximum headway of 1 and 3 minutes, respectively. The cost of the arcs (in blue) is computed using as simple penalty function the absolute value of the deviation from the ideal headway, in seconds. The minimum cost feasible O_d^--O_d^+ path in the graph correponds to the timetable 7:01–7:03–7:05, that has a cost of 0, which means that the optimal total deviation from the ideal headways is 0, as it is immediate to verify.

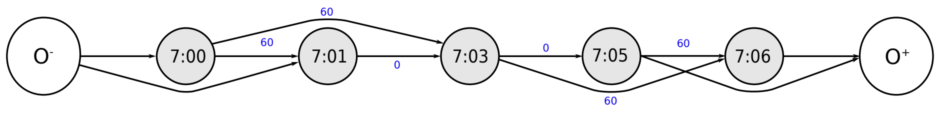

Figure 2: G_d^{TT} compatibility graph.

4.2 VS model

The VS model is based on representing feasible vehicle schedules in terms of flows on a *single* directed *VS graph* $G^{VS} = (\, N^{VS} \,,\, A^{VS} \,)$, which is also basically a compatibility graph: the VS sub-problem is not separable per direction, because the schedule for each single vehicle can–and usually does—cover trips belonging to different directions (*interlining*). Using compatibility graphs to represent VS is well-known in the literature [3]; however, in a standard VS, such as when the problem is solved in the classical planning sequence, one typically has to construct feasible vehicle schedules that cover all the trips of some *input* timetable π^*. In our ITTVS, instead, the optimal timetable π^* is unknown (being part of the integrated decision), hence the VS sub-problem feasible space consists of all feasible vehicle schedules that can be constructed from the whole input set of trips $\mathcal{T}$. Indeed, a vehicle schedule is, from a combinatorial point of view, a *sequence* of trips such that two subsequent ones are compatible according to the given VS constraints.

In the following, we will actually present *two* different VS graphs, which attain different trade-offs between $|N^{VS}|$ and $|A^{VS}|$ (basically, the number of linear constraints and variables in the corresponding MILP sub-model). Common to both versions is that N^{VS} contains *two* nodes i^- and i^+ for each trip $i \in \mathcal{T}$, representing the start and the end of trip i, respectively.

"Pure" compatibility graph. In the first variant, besides the previously mentioned trip beginning and ending nodes, G^{VS} only contain two further nodes O^- and O^+, whose out-coming and in-coming arcs respectively represent a vehicle performing the first pull-out and the last pull-in trips of the corresponding schedule. As for A^{VS}, it contains six types of arcs:

1. *Trip arcs* (i^-, i^+) for each trip $i \in \mathcal{T}$ (red arcs in Figure 3), with capacity 1 and 0 cost; a unit of flow on the arc means that the corresponding trip i is "covered" in the vehicle schedule.

2. *In-line compatibility arcs* (i^+, j^-) for each pair of trips i and j that are in-line compatible (blue arcs in Figure 3); a unit flow on the arc means that the bus covering trip i will be waiting at the terminal $en(i) = sn(j)$ and then start trip j. These arcs have capacity 1 and cost proportional to the *extra waiting time* $st(j) - et(i) - \delta^{h(i)}_{min,en(i)}$ w.r.t. the minimum waiting time at $en(i)$ in the given time window.

3. *Out-line compatibility arcs* (i^+, j^-) for each pair of trips i, j that are out-line compatible (green arcs in Figure 3); a unit of flow on the arc means that the vehicle covering trip i will perform a pull-in trip from $en(i)$ in time window $h(i)$, then perform a pull-out trip to the $sn(j)$ in time window $h(j) \geq h(i)$, then finally start performing trip j. These arcs have capacity 1 and cost proportional to $t^{h(i)}_{en(i)+} + t^{h(j)}_{sn(j)-}$, i.e., the sum of the *pull-in/out travel times* in the corresponding time windows.

4. *Start arcs* (O^-, i^-) for each trip $i \in \mathcal{T}$ (dotted arcs in Figure 3); a unit of flow on the arc means that a vehicle will perform a pull-out trip to $sn(i)$ as the first activity of its vehicle block. These arcs have capacity 1 and cost proportional to the pull-out time $t^{h(i)}_{sn(i)-}$.

5. *End arcs* (i^+, O^+) for each trip $i \in \mathcal{T}$ (also dotted arcs in Figure 3); a unit of flow on the arc means that a vehicle will perform a pull-in trip to return to the deposit from $en(i)$ as the last activity of its vehicle block. These arcs have capacity 1 and cost proportional to the pull-in time $t^{h(i)}_{en(i)+}$.

6. *Return arc*, the single (O^+, O^-) (omitted in Figure 3). This is added in order to allow any number of units of flow, i.e., vehicles, to depart from O^- and reach O^+, thereby being used to define the vehicle schedule. By setting all *deficits* of the nodes to 0, this defines a *circulation problem* on the VS graph. The arc has capacity equal to the maximum fleet cardinality (if any, $+\infty$ otherwise) and a "large" cost (w.r.t. those that can typically be expected on the other types) representing the cost of using one more vehicle in the vehicle schedule.

Example 2. Consider the set $\mathcal{T}$ formed of the 5 trips described in Table 2. For simplicity, each trip lasts 90 minutes, travel times from/to the deposit to/from both terminals are all equal to 15 minutes, and all minimum stopping times are 30 minutes. In the table, for each trip i we report the corresponding direction (i.e., either $\vec{AB}$ or $\vec{BA}$), its start and end time $st(i)$ and $et(i)$, and the *start/end depot time* instants $sd(i)$ and $ed(i)$. These are respectively the last instant in which a vehicle can start a pull-out trip in time to reach $sn(i)$ and perform the trip i ($sd(i) = st(i) - t^{h(i)}_{sn(i)-}$), and the first instant in which a vehicle, after having performed a pull-in trip from $en(i)$ at the end of trip i, is ready to leave again the deposit ($ed(i) = et(i) + t^{h(i)}_{en(i)+} + \delta^{h(i)}_{min,O}$). Figure 3 shows the "pure" compatibility graph for the example.

i	direction	$st(i)$	$et(i)$	$sd(i)$	$ed(i)$
1	$\vec{AB}$	7:00	8:30	6:45	9:15
2	$\vec{BA}$	9:00	10:30	8:45	11:15
3	$\vec{AB}$	11:00	12:30	10:45	13:15
4	$\vec{BA}$	13:00	14:30	12:45	15:15
5	$\vec{AB}$	15:00	16:30	14:45	17:15

Table 1: A VS example.

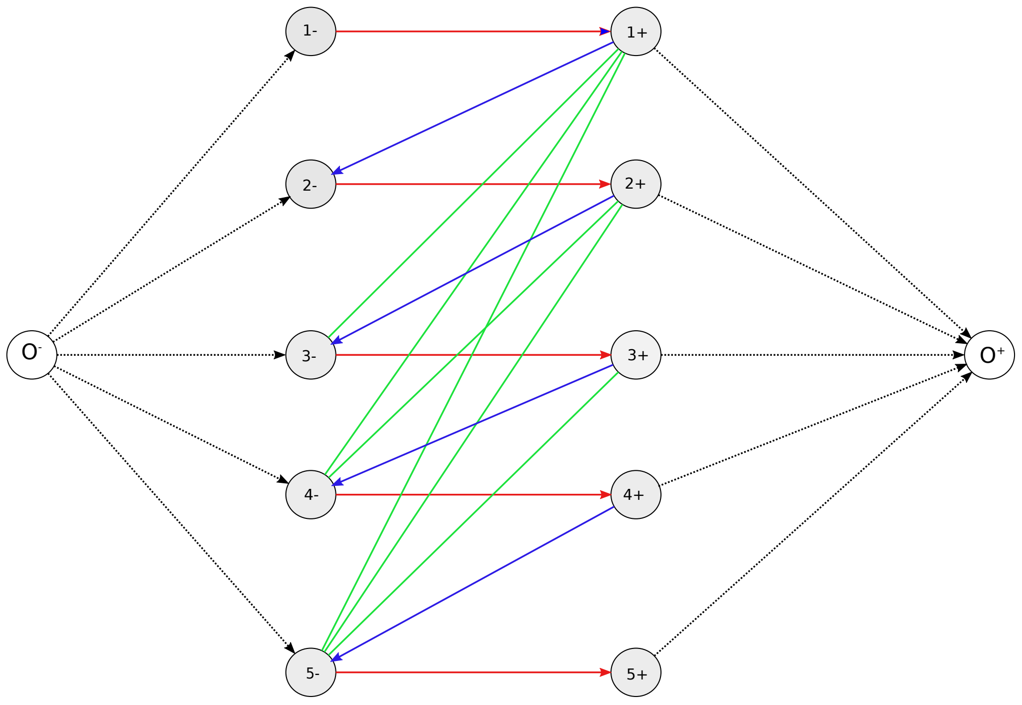

Figure 3: G^{VS} “pure” compatibility graph for the example.

The issue with this variant of G^{VS} is that it has a rather large number of out-compatibility arcs. Indeed, if a trip ends rather early (with respect to the planning horizon), it is likely to be out-compatible with most of the subsequent trips. We can reduce the number of arcs constructing an alternative VS graph as follows.

Compatibility/time-space graph. To remove all the out-compatibility arcs, we can introduce “time-depot” nodes O_t for properly chosen time instants t. In particular, for each trip $i \in \mathcal{T}$ we will define the *start-time-depot* $O_{sd(i)}$ and *end-time-depot* $O_{ed(i)}$, with the start/end depot time instants $sd(i)$ and $ed(i)$ having been defined in Example 2. We denote by $\bar{T}$ the set of time instants corresponding to all the start/end time-depot nodes in G^{VS}. Next, after having removed the out-line compatibility arcs we add the following arcs:

- *Time arcs* (O_t, O_{t+1}) for all pairs $(t, t+1)$ of time instants in $\bar{T}$, where $t+1 = \min\{ t' \in \bar{T} : t' > t \}$ (vertical green arcs in Figure 4). These are the typical “holding arcs” in time-space graphs, representing the fact that all vehicles at the deposit at t that have not just started a pull-out trip will remain at the deposit until $t+1$. The cost of these arcs is 0 and the capacity is set equal to the maximum fleet cardinality (if any, $+\infty$ otherwise).
- *Pull-in arcs* $(i^+, O_{ed(i)})$ for all $i \in \mathcal{T}$ (diagonal green arcs in Figure 4), representing the fact that the vehicle having just performed trip i performs a pull-in to the deposit, where it arrives at time $ed(i)$. These arcs have capacity 1 and cost proportional to the pull-in time $t^{h(i)}_{en(i)+}$.

- *Pull-out arcs* $(O_{sd(i)}, i^-)$ for all $i \in \mathcal{T}$ (diagonal green arcs in Figure 4), representing the fact that the vehicle performs a pull-out trip at time $sd(i)$, i.e., just in time to subsequently start trip i. These arcs have capacity 1 and cost proportional to the pull-out time $t^{h(i)}_{sn(i)-}$.

Basically, in this version the depot nodes O^- and O^+ are expanded in a space-time (line) graph representing the status of the depot (number of vehicles available there) at all possible times where it may change; this is why we dub it a "compatibility/time-space graph". In this version, each out-compatibility arc between two trips i and j is replaced by the i^+–j^- path consisting of a pull-in arc from i^+ to $O_{ed(i)}$, a sequence of time-arcs connecting the time-depot nodes $O_{ed(i)}$ and $O_{sd(j)}$, and a pull-out arc from $O_{sd(j)}$ to j^-. The advantage of this version is that of replacing the potentially $O(|\mathcal{T}|^2)$ out-line compatibility arcs with $O(|\mathcal{T}|)$ new nodes and arcs. Note that for some $i \neq j$, one may have $ed(i) = sd(j)$, which means that there may be strictly less than $2|\mathcal{T}|$ nodes O_t (and that, unlike in Figure 4, these nodes can have more than three incident arcs). In our experiments, the compatibility/time-space graph has usually outperformed the "pure" compatibility one. As a final remark, if deadheading is allowed, then *deadhead arcs* must be added to A^{VS} (in either version) that are analogous to out-line compatibility arcs save for not contemplating a return to the depot. This may seem to run contrary to the rationale of the compatibility/time-space graph, but in practice deadheading is likely to be only possible for relatively few pairs of trips for which $en(i)/sn(j)$ and/or $et(i)/st(j)$ are "rather near", in that otherwise it is more reasonable (or required by regulations) to return to the depot anyway. Thus, the compatibility/time-space graph may also be a sensible choice in the presence of deadheading.

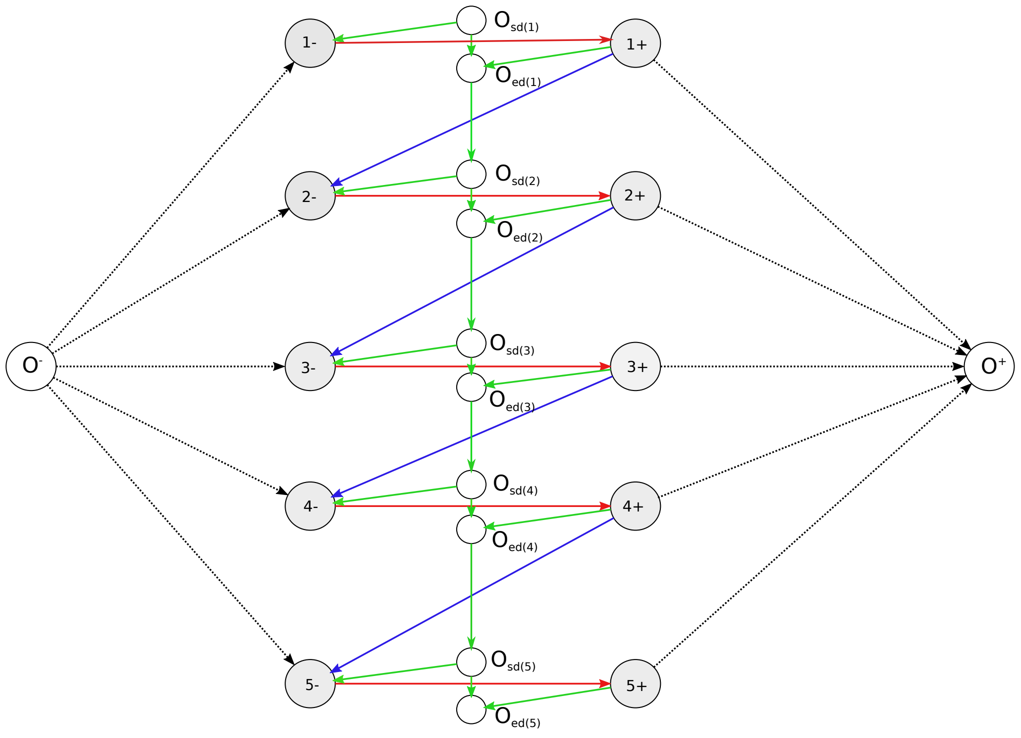

Figure 4: G^{VS} compatibility/time-space graph.

Whatever the chosen version, if the VS subproblem could be solved independently—and it can not—then it would be a min-cost circulation problem, i.e., a *minimum cost network flow (MCF)* on G^{VS}, which is polynomially solvable. Actually, the optimal solution to the VS sub-problem would trivially be the *zero circulation*, as the arc costs are non-negative and the node deficits are zero. In fact, it is due to the linking constraints described in the next sub-section, that flow will be forced to traverse the trip-arcs corresponding to the trips $\mathcal{T}^*$ selected by the TT sub-problems, and therefore produce a non-zero circulation (vehicle schedule Ω^*).

It should be remarked that the VS subproblem discussed in this section may not be capable of expressing some constraints on the vehicle routes that may be necessary in certain cases, such as those depending on the total time/distance travelled by the vehicle (refuelling, cleaning, servicing, ...). Yet, some constraints on the vehicle schedules can indeed be represented by appropriate modifications to either the TT graphs or the VS graph; examples are provided in the next Section 5. Furthermore, it is in principle possible to replace the graph-based VS model with any more expressive one, e.g. based on set partitioning formulations, without changing the overall structure of the integrated model (except, very possibly, making the ITTVS even more difficult to solve in practice).

4.3 TT-VS integrated model

The integrated model combines the VS graph (in whatever version) and the TT graphs for all directions $d \in \mathcal{D}$ to yield the following MILP model:

$$\min \alpha c x + \sum_{d \in \mathcal{D}} c^d y^d \tag{1}$$

$$\sum_{(m,n) \in A_d^{TT}} y^d_{m,n} - \sum_{(n,m) \in A_d^{TT}} y^d_{n,m} = b^d_n \qquad n \in N_d^{TT}\ ,\ d \in \mathcal{D} \tag{2}$$

$$y^d_{n,m} \in \{0,1\} \qquad (n,m) \in A_d^{TT}\ ,\ d \in \mathcal{D} \tag{3}$$

$$\sum_{(m,n) \in A^{VS}} x_{m,n} - \sum_{(n,m) \in A^{VS}} x_{n,m} = 0 \qquad n \in N^{VS} \tag{4}$$

$$0 \leq x_{n,m} \leq u_{n,m} \qquad (n,m) \in A^{VS} \tag{5}$$

$$\sum_{(n,m) \in B(i)} y^{d(i)}_{n,m} \ = \ x_{i^-,i^+} \qquad i \in \mathcal{T} \tag{6}$$

The MILP formulation is clearly formed of three distinct blocks. Constraints (2) are the flow conservation constraints of the TT subproblems, where $y^d_{n,m}$ are the arc flow variables on A_d^{TT}; the deficits b^d_n for TT are all 0 except for $n \in \{\, O_d^- \,,\, O_d^+ \,\}$, which, together with (3), ensures that the solution describes a path between O_d^- and O_d^+ in G_d^{TT} (timetable π_d). Similarly, $x_{n,m}$ are the arc flow variables on A^{VS}, and (4) the corresponding flow conservation constraints describing a circulation (vehicle schedule Ω) in G^{VS}. The capacities $u_{n,m}$ are all 1 except that of the return arc (O^+, O^-) and the time arcs (if any); the variables need not be declared integer, once this is done for the y^d, due to the total unimodularity of flow constraints. Finally, (6) are the *linking constraints* ensuring that a trip is performed in the VS if and only if it is chosen by the corresponding TT; $B(i)$ is the backward star of the node of $N_{d(i)}^{TT}$ corresponding to trip i, i.e., the set of all arcs in $A_{d(i)}^{TT}$ entering it.

The bi-objective nature of the integrated TT-VS problem is modeled using the *weighted objective function* (1), where the VS objective is scaled by a coefficient α representing the decision-maker preferences in terms of priority between the two objectives. As already remarked, the VS costs c already are obtained by properly weighting one main VS objective with two minor ones, which means that constructing (1) requires properly choosing no less than three scaling parameters. Of course, the main one is α, governing the compromise between the two contrasting objective functions of the problem (service quality vs. service provider cost). The experience of *MAIOR* personnel has been instrumental in properly setting these weights.

5 Extensions

We now describe two extensions of the models presented in the previous Section that allow to deal with nontrivial constraints on either the TT or the VS by properly modifying the underlying graphs. This is done primarily to show the flexibility of our approach and its capacity to be adapted to the different needs of different service providers, which is one of the defining technical capabilities that makes *MAIOR* a global player in its market.

5.1 Complex lines

The first extension that we consider is that of a *complex single line*, which has two *sets* of terminals $\mathcal{A}$ and $\mathcal{B}$; each trip has the form either $\overrightarrow{A_i B_j}$ or $\overrightarrow{B_j A_i}$ for $A_i \in \mathcal{A}$ and $B_j \in \mathcal{B}$. Thus, each direction of the line is actually composed of more than one *pattern*, corresponding to different choices of the terminals. Crucially, all the patterns have to share a common central segment, where the main-stops are located, as depicted in Figure 5 for the so-called "double Y" topology. Indeed, the headway for a direction is computed as the time separating two consecutive trips i and j at the main-stop running in that direction, *irrespectively from the pattern* they belong to, i.e., from the specific starting terminal in $\mathcal{A}$ and ending one in $\mathcal{B}$. Accounting for this case is actually simple enough provided that trips from different pairs of terminals *follow a regular scheme*. Indeed, for a complex line, service providers typically specify how the trips of the line alternate between different pairs of terminals, for instance by a simple *departure sequence scheme* σ^d and *arrival sequence scheme* σ^a, as illustrated in the following example.

Example 3. Consider the double Y topology, shown in Figure 5, with departure scheme $\sigma^d = (\, A_1 \,,\, A_2 \,)$ and arrival scheme $\sigma^a = (\, B_1 \,,\, B_1 \,,\, B_2 \,)$. This means that trips must alternate first one vehicle departing from A_1 and then one departing from A_2, and a vehicle arriving from B_2 after two consecutive ones arriving from B_1. Since the departure scheme has length 2, while the arrival scheme has length 3, the complete departure-arrival sequence scheme $\sigma = (\, A_1\text{-}B_1 \,,\, A_2\text{-}B_1 \,,\, A_1\text{-}B_2 \,,\, A_2\text{-}B_1 \,,\, A_1\text{-}B_1 \,,\, A_2\text{-}B_2 \,)$ has length 6, and repeats indefinitely along all the planning horizon.

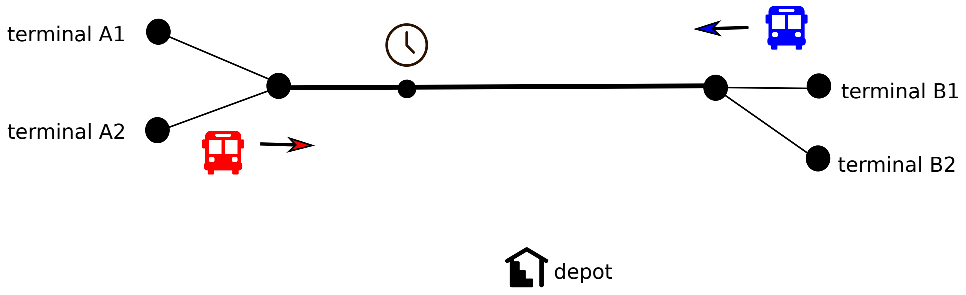

Figure 5: "Double Y" complex single line.

Therefore, it is only necessary to modify the TT subproblem to account for the fact that the trips have to follow the scheme σ. In fact, nodes of G_d^{TT} are trips, i.e., pairs of terminals; arcs can therefore be seen as having the general form $(\, sn(i)\text{-}en(i) \,,\, sn(j)\text{-}en(j) \,)$ (although, of course, the time of the trip also plays a role). Ensuring that the chosen path follows the right sequence basically only amounts at removing compatibility arcs between nodes that violate it, although they would be feasible in terms of the corresponding headway. However, this would work on the original G_d^{TT} only if each *trip type* (oriented pair of terminals) appeared at most once in σ; yet, as our example shows, this is not necessarily true. Hence, one also needs to keep track of the *position in* σ of the current node $(\, sn(i)\text{-}en(i) \,)$, in order to construct the correct compatibility arcs. This can be done by replicating it for each of its occurrences in σ, as the example below further illustrates.

Example 3 (continued)**.** In our example, $(\, A_1\text{-}B_1 \,)$ appears twice in σ, so we need to replicate all the nodes of this type twice to recognise whether it is the first or the second occurrence of $(\, A_1\text{-}B_1 \,)$ in σ. The same holds for $(\, A_2\text{-}B_1 \,)$. A slice of the corresponding modified compatibility graph G_d^{TT} is shown in Figure 6, where the duplicated nodes (trips) are highlighted in red; the superscript indicates the position in the sequence.

Of course, this comes at a cost of a possibly considerable increase in the number of nodes of G_d^{TT}, although the arcs do not grow quite as rapidly because the construction is precisely aimed at removing those arcs that do not follow the right order. However, the length of σ, and therefore the size of G_d^{TT}, may grow rather rapidly as $|\mathcal{A}|$ and $|\mathcal{B}|$, and/or the complexity of σ^d and σ^a, increase. Yet, there are not too many different complex line topologies that appear in practice (the "double Y" being almost, but not quite, the most complex that can reasonably happen), nor there is usually reason to have particularly complex schemes σ^d and σ^a.

In all this, the VS model is completely unaffected. We will next present an "orthogonal" modification that rather involves G^{VS} only, leaving the G_d^{TT} unchanged.

5.2 Vehicle flow control

Public transport planners are often able to accurately estimate the number of vehicles required for different *periods* of the planning horizon, which may or may not coincide with the time-windows defined in the

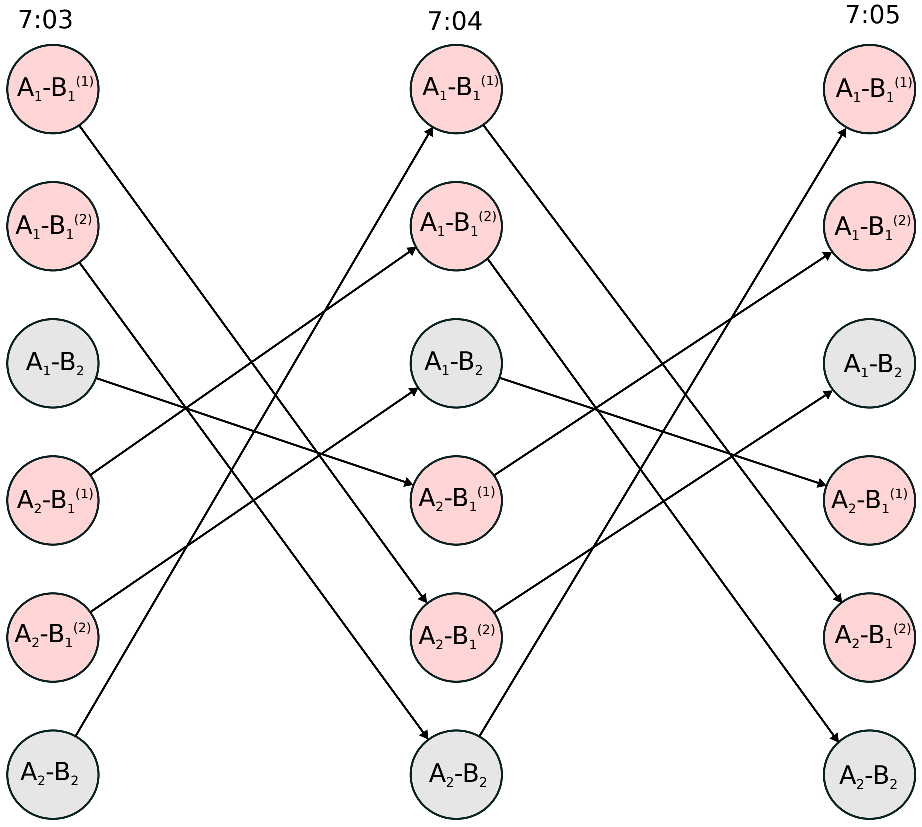

Figure 6: A slice of the modified G_d^{TT} graph for the scheme σ.

previous sections. These estimates can be so accurate that the planner may want to *fix the number of vehicles per period* on input. This is relatively easy to do by modifying the graph G^{VS}, in particular in its compatibility/time-space variant (cf. Section 4.2); indeed, as we already observed, the capacity of specific arcs in G^{VS} can be used to set a limit on the number of vehicles, so it is not hard to bring this idea further and actually *fix* the actual number of vehicles by, basically, *fixing* the flow on some arcs. More specifically, for each of the periods $h = 1, \ldots, r$, we define $\phi(h)$ to be the number of vehicles to be fixed, and we add to N^{VS} a *local source node* O_h^- and a *local sink* node O_h^+. These nodes are given deficits that depend on the number of vehicles estimated for the corresponding period and the following/preceding period (with the right sign), as detailed below:

- for O_1^-, $-\phi(1)$, i.e., (minus) the desired number of vehicles for the first period;
- for O_h^-, $-\max\{\, 0\, ,\, \phi(h) - \phi(h-1)\, \}$, for $h = 2, \ldots, r$, i.e., (minus) the difference between the number of vehicles circulating in period h and those circulating in period $h - 1$ if this is positive, i.e., new vehicles have to enter in period h;
- for O_h^+, $\max\{\, 0\, ,\, \phi(h) - \phi(h+1)\, \}$, for $h = 1, \ldots, r-1$, i.e., the difference between the number of vehicles circulating in period $h+1$ and those circulating in period h if this is negative, i.e., vehicles have to leave after the end of period h;
- for O_r^+, $\phi(r)$, i.e., the desired number of vehicles for the last period.

This means that the VS sub-problem is no longer a circulation one, since it has as many source/sink pairs as there are periods (the deficit of all other nodes remains 0), and in fact the return time arc is also removed. Finally, arcs (O_i^-, i^-) and (i^+, O_h^+) (with 0 cost and unitary capacity) are added for all trips i belonging to period h. This construction is illustrated in Figure 7 for the same fragment of the (compatibility/time-space) graph G^{VS} of Figure 4, assuming there are two periods 7:00–12:30 and 12:30–16:00 in input with 1 and 2 vehicles *fixed*, respectively.

A benefit of this construction is that the primary VS cost component (i.e., the number of vehicles) is

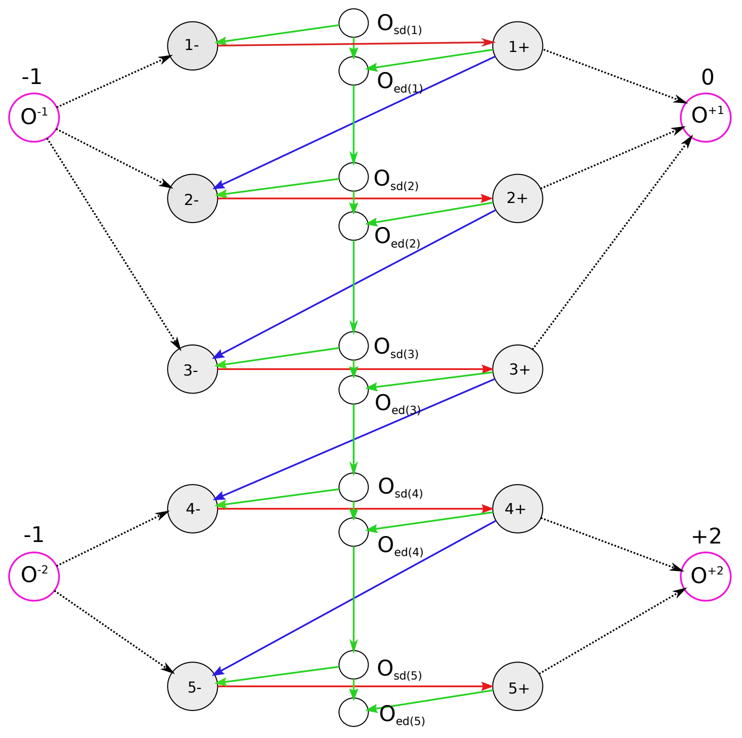

Figure 7: Modified G^{VS} graph with vehicle flow control

now fixed, since the number of vehicles is so. Indeed, in G^{VS} the cost was associated to the return time arc, which has now disappeared. In practice, it has been observed that this may make it somewhat easier to find good values for the weighting parameter α in the objective function.

6 Solution Approach

Real-life MILP instances of ITTVS as defined in the previous section are too hard to be solved directly using a general-purpose solver within the time constraints dictated by the planners' operational requirements. Therefore, a heuristic approach is needed. We now describe a *matheuristic*, called `TTD` (*TimeTabling Design*), based on the solution of the continuous relaxation of (1)–(6) and on a classic *diving approach*, that at each iteration fixes "some" trips and progressively constructs a feasible ITTVS solution. The fixing is basically greedy, in that decisions taken at one iteration are usually not challenged in subsequent ones, although a very limited amount of backtracking is allowed when infeasibility of the choices is detected. The iterative process runs until a complete integer solution is obtained, which basically means that for all $d \in \mathcal{D}$ the corresponding π_d forms a complete O_d^--O_d^+ path, or infeasibility is detected that the backtracking is not able to resolve.

The two relevant technical aspects of the approach are how fixing is performed, and how the solution of the continuous relaxation is computed; these are basically orthogonal, and therefore are separately discussed.

6.1 Fixing strategy

The fixing strategy is based on the value of a continuous solution $(\tilde{x}, \tilde{y} = [\tilde{y}^d]_{d\in\mathcal{D}})$, irrespectively on how this has been computed. Fixing is clearly the crucial aspect of a diving heuristic, and extensive experiments were necessary to find a fixing strategy that is both robust and efficient in practice. The best performing fixing strategy among the ones we tested turned out to be the so called *V-fix* one. On the outset, the idea is simple: to fix the trips, we first sort them in descending order of the continuous solution value $(\tilde{x}, \tilde{y})$. However, note that, in our model, each trip $i \in \mathcal{T}$ is associated to *two* continuous solution values: $\tilde{y}_i$ in the corresponding TT, and $\tilde{x}_{(i^-, i^+)}$ in the VS. Which of the two is chosen depends on the particular *stage* the fixing rule is in, as detailed below. Indeed, a crucial component for the effectiveness of the fixing rule is to carefully restrict the set of trips that we consider as candidates for being fixed depending on the previous history, as we now detail. For simplicity, the description is limited to the simple single line case, and we also assume to know that the very first trip has to start from terminal A going towards B, rather than vice-versa, which is usually quite clear to planners. However, the fixing rule can be extended to complex single lines and beyond.

At the first iteration, when no trips are fixed, we select the direction $d = \vec{AB}$, and we restrict the set of candidates to the trips in the forward star of the corresponding dummy source node O_d^- in the corresponding TT subgraph G_d^{TT}, ordering them in terms of the values of the corresponding $\tilde{y}_i^d$. At the second iteration, with just one trip $\bar{i}$ fixed, we rather select the opposite direction $d = \vec{BA}$ and we restrict the set of candidates to the trips in the forward star of node $\bar{i}^+$ in the VS graph, i.e., the node representing the possible ways to chose an activity for the vehicle having just performed trip i (an in-line compatible trip, an out-line compatible trip, or a return to the depot), ordered by the corresponding $\tilde{x}_{(i^-,i^+)}$ variable instead. These two initial fixings form a "V" in the time-space graph used by transport planners to represent a timetable, whence the name. In the subsequent iterations, we restrict the set of candidates to the union of the trips in both the forward star and the backward star in the TT graphs of the trips that have been fixed, sorted in descending order of the corresponding $\tilde{y}_i$. We then proceed at fixing the one with highest value, provided that a reasonable balance is kept between the number of trips fixed for each direction. That is, if the difference is less than 20% we allow selecting the trip to be fixed irrespectively of the direction d to which it belongs, otherwise we only select trips for the direction with fewer fixings.

In order to further reduce the problem size, when we fix a trip i to one (as belonging to the solution), we also fix to zero (as to not belonging to the solution) the trips belonging to the "neighborhood" of i that would violate the TT headway if they were selected, because they are "too close" in time to i. After all these fixings, the problem may have become unfeasible; before confirming the fixing we check that this has not happened by solving the corresponding TT and VS subproblems, and in case we backtrack on the decision and move to the next candidate in the list. If the list becomes empty, without any (locally) feasible fixing having been identified, we accept failure and we exit from the heuristic (although this has never happened in practice, after that the fixing rule has been properly tuned).

6.2 Continuous Solution

We consider two ways to find a continuous solution for (1)–(6): a general-purpose LP solver and a Lagrangian relaxation. In the former case, we relax the integrality constraints (3) and solve the corresponding LP using `Clp` (Coin-OR linear programming) [10], an open-source LP solver written in `C++`. Implementing the fixing in this case is trivial by just changing the bounds on the affected variables. Besides fixing to 1 the lower bound of variables corresponding to trips that are chosen to be a part of the current partial solution, we also fix to 0 the upper bound of variables representing trips that cannot possibly be chosen together with that, as discussed above.

The alternative is to use Lagrangian techniques. This corresponds to defining the vector of *Lagrangian multipliers* $\lambda = [\,\lambda_i\,]_{i\in\mathcal{T}}$ associated to the *linking constraints* (6), and solve the corresponding *Lagrangian relaxation*

$$P(\lambda) \quad \min\Big\{\, \alpha cx + \sum_{d\in\mathcal{D}} c^d y^d + \sum_{i\in\mathcal{T}} \lambda_i \Big[\sum_{(n,m)\in B(i)} y^{d(i)}_{n,m} - x_{i^-,i^+} \Big] \;:\; (2)-(5) \,\Big\} \;.$$

Clearly, $P(\lambda)$ actually consists of $|\mathcal{D}| + 1$ independent sub-problems, a $TT_d(\lambda)$ for each $d \in \mathcal{D}$ and a single $VS(\lambda)$, that can be solved separately, respectively, as acyclic SPs and a MCF on the corresponding graphs G_d^{TT} and G^{VS}, as discussed on the previous sections. Note that the integrality constraints (3) are not an issue since all the individual sub-problems have the integrality property; thus, for any choice of λ, the corresponding Lagrangian relaxation solution is integral. It is well-known that for each choice of λ, $P(\lambda)$ is a relaxation of $ITTVS$, i.e., $\nu(P(\lambda)) \leq \nu(ITTVS)$ ($\nu(\cdot)$ denoting the optimal value of an optimization problem). To find the best possible Lagrangian relaxation, one then has to solve the *Lagrangian Dual*, i.e., maximize the *Lagrangian function* $\nu(P(\lambda))$ over all $\lambda \in \mathbb{R}^{|\mathcal{T}|}$. Even with the best possible choice λ^* of the Lagrangian multipliers, there is no guarantee that the penalty term in the objective function will lead to a feasible integer solution in $P(\lambda^*)$, i.e., one that satisfies the linking constraints (6). However, it is well-known that the Lagrangian Dual is equivalent to the *convexified* *relaxation* of the original problem; since in our case all the subproblems have the integrality property, this is actually the same as the continuous relaxation [17] of ITTVS, as solved by the previous approach. Thus, the two approaches provide the same lower bound, and thereby arguably continuous solution of "the same quality". However, this does not mean that the time required to find the continuous solution is the same, and that the solutions themselves are necessarily identical.

Both aspects (time and solution) obviously depend on the specific algorithm used to solve the Lagrangian dual; in our case we are using an implementation of the proximal Bundle approach already used with success in other applications (e.g., [19, 18]). Besides finding the optimal Lagrangian multipliers vector λ^*, the Bundle method also allows to explicitly construct the optimal primal solution $(\tilde{x}, \tilde{y})$. Technically, this is done by collecting the (integer, unfeasible) primal solutions generated at each iteration, out of which the (continuous, feasible) $(\tilde{x}, \tilde{y})$ is generated via *convex multipliers* that are automatically produced by the *master problem* solved at each iteration [16, 17]. Although the exact details are well-known and not worth reporting here, it can be useful to remark that while a *feasible* $(\tilde{x}, \tilde{y})$ is only produced at the last iteration, the algorithm produces an *unfeasible* primal solution at each iteration, and (roughly speaking) the "degree of unfeasibility" quickly decreases as the algorithm proceeds. Clearly, for the purpose of driving the fixing strategy, which is the only use of $(\tilde{x}, \tilde{y})$ in our setting, there is no strong requirement that the solution be feasible. In this sense, our diving heuristic can be considered as well a *Lagrangian heuristic*, where the solutions of the Lagrangian subproblems are used to guide the construction of a feasible solution for the original problem; the use of the (not necessarily feasible) "convexified" primal solution in this context has already been shown to be effective in several applications [4]. All this allows us to explore the trade-off between terminating the algorithm early, thereby settling for a less "exact" $(\tilde{x}, \tilde{y})$ and lower bound, but gaining in solution time, or letting it run to termination, thereby obtaining a solution $(\tilde{x}, \tilde{y})$ with the same quality as that produced by `Clp` (albeit not necessarily exactly the same one).

Besides the approach for (iteratively) generating the Lagrangian multipliers λ, it is of course relevant how the subproblems are solved. For the TT subproblems, an hand-made implementation of the classical acyclic Shortest Path algorithm on the TT graphs G_d^{TT} is used. As for the VS subproblem, the general-purpose MCF solver `MCFSimplex` from the `MCFClass` project [26] (based, as the name suggest, on the network simplex algorithm) has been used.

A final, but important detail of the Lagrangian approach concerns how fixing is enforced in the subproblems. This is nontrivial, as several problems for which polynomial algorithms exist, among which notably shortest path ones, easily become $\mathcal{NP}$-hard if specific features are required from the solution. Fortunately, in our case this is not an issue. Indeed, for the TT subproblems we can easily fix a trip (node) as necessarily belonging to the chosen O_d^--O_d^+ path by exploiting the fact that G_d^{TT} is acyclic: it is sufficient to remove all the arcs in A_d^{TT} overstepping it. Of course, fixing to zero (removing) a trip is easily obtained by removing from A_d^{TT} all arcs entering it. As for the VS subproblem, fixing to 1 a trip arc (i^-, i^+) corresponds to setting a deficit of ± 1 (with appropriately chosen signs) on its end-nodes and removing it, while fixing it to 0 just amounts at removing it (or, equivalently, setting $u_{i^-,i^+} = 0$).

7 Testing

To illustrate the results of our approach we selected 12 *real-world* bus lines covering the city center of 3 major Italian cities, among which Milan. This data has been provided by the corresponding bus service providers, all *MAIOR* customers. All the tests are on *single lines*, albeit possibly "complex" ones (cf. Section 5.1). Table 2 summarises the main characteristics of the instances, that are of three different types: (*i*) simple (A-B) having $|N| = 2$ terminals, (*ii*) complex "Y" (A-B_1B_2) with $|N| = 3$ terminals, and (*iii*) complex "double-Y" (A_1A_2-B_1B_2) with $|N| = 4$ terminals. All simple topologies and two of the "Y" topologies have no frequency schemes, one of the "Y" topologies has $|\sigma| = 4$, and the "double-Y" one has $|\sigma| = 6$. Time is discretized in minutes, and for each minute of the time horizon there is a possible trip i in $\mathcal{T}$ for each pattern. A typical time-horizon ranges roughly from 5:00 to 24:00, for a total of 1140 possible trips for each pattern. There are usually around 9 time windows, or about one time window each two hours. In Table 2 we report, for each line, the time horizon, the number of terminals, and the *average over the different time-windows* of ideal headways, min/max dwell times, and pull-in/out times. Recall that there is no maximum dwell time for the depot. The length of the lines, not reported in the table as it is not part of the algorithm input, ranges from 4 to 16 Km, with an average of 11 Km.

instance	T (hh:mm)	$\|N\|$	I^h (mm:ss)	$\delta^h_{n,min}$ (m)	$\delta^h_{n,max}$ (m)	$t^h_{n\pm}$ (m)
`2Cap_R1`	05:30 24:00	2	11:00	3	10	20
`2Cap_R2`	05:30 24:00	2	12:30	3	30	21
`2Cap_R3`	05:00 24:00	2	12:00	3	10	20
`2Cap_F1`	05:15 22:50	2	13:30	2	20	18
`2Cap_F2`	05:50 20:40	2	23:00	2	10	10
`2Cap_F3`	05:20 24:20	2	18:00	2	15	9
`2Cap_F4`	07:00 20:25	2	09:00	2	10	17
`3Cap_F5`	05:45 24:08	3	08:00	2	15	18
`3Cap_F6`	06:34 20:53	3	07:30	2	10	18
`3Cap_F7`	05:24 22:44	3	08:00	2	18	17
`3Cap_M1`	05:57 20:31	3	11:00	2	13	13
`4Cap_F8`	06:40 20:24	4	06:00	2	17	16

Table 2: Instance data.

7.1 First case study: simple and complex lines

We first report results for the unabridged version of our matheuristic, i.e., without vehicle flow control (but, necessarily, with the handling of "complex" lines). We consider two versions of our heuristic approach, as described in Section 6: "h-B" (approximately) solves the Lagrangian dual using the Bundle method, "h-C" solves the continuous relaxation of (1)–(6) using the open-source LP solver `Clp`. All the experiments have been performed on a PC with a 1.9 Ghz Intel Xeon (R) E5-2420 processor. The MILP models were solved by the commercial solver `Cplex 12.7`. For the LP relaxations we used the `Clp` solver included in `Cbc-2.9.8`, in particular we applied the *barrier* method at the first iteration, and the *dual simplex* in the subsequent ones. Finally, the stopping criterion used for the `Bundle` method is a maximum number of iterations, and for all our test instances it never reached the optimal value. All the solvers were finely tuned by performing extensive experiments in order to find a good trade-off between solution quality and running time.

In Table 3 we compare the performance of the two versions in terms of: (*i*) *lower bounds* obtained at the very first iteration, before any fixing is done (`Clp` computes the exact optimal value of the continuous relaxation of (1)–(6), whereas the Bundle only computes a lower approximation due to being stopped early); *upper bounds* (value of the feasible ITTVS solution produced), and (*iii*) solution *time* (in minutes). To improve readability, we often report the *percentage difference* between two values X *vs* Y, denoted by $\phi_{\%}(X,Y)$, computed as $\phi_{\%}(X,Y) = (X - Y)/Y \times 100$.

	h-C	h-B	h-B *vs.* h-C		
instance	time	time	time $\phi_{\%}$	LB $\phi_{\%}$	UB $\phi_{\%}$
`2Cap_R1`	6	28	393.82	-4.28	-3.53
`2Cap_R2`	13	36	167.72	-2.64	-5.81
`2Cap_R3`	8	28	249.04	-6.76	-0.93
`2Cap_F1`	10	23	124.05	-5.07	-2.31
`2Cap_F2`	3	6	86.14	-5.57	-0.41
`2Cap_F3`	13	19	49.28	-9.75	-30.17
`2Cap_F4`	2	16	842.01	-2.72	-4.64
`3Cap_F5`	116	172	48.73	-18.32	-4.39
`3Cap_F6`	47	101	114.57	-2.56	-3.58
`3Cap_F7`	628	195	-68.89	-6.34	10.60
`3Cap_M1`	66	53	-19.74	-4.01	0.46
`4Cap_F8`	2410	438	-81.85	-7.96	4.37

Table 3: h-B *vs* h-C, without vehicle flow control.

Table 3 shows that h-C is generally faster than h-B, apart from the last three (larger, and more difficult) instances. The lower bounds computed by the Lagrangian approach are uniformly (and sometimes consistently) weaker, for two reasons: the maximum limit on the number of iterations of the Bundle algorithm, and the fact that `Clp` generates cutting planes to strengthen the continuous relaxation of the MILP (1)–(6). Finally, the quality of the feasible ITTVS solutions found by h-B is usually better, apart from the last three instances, suggesting that the $(\tilde{x}, \tilde{y})$ solution computed by the Lagrangian approach, although not feasible, is often more "informative" than the standard continuous relaxation solution. The fact that this does not happen with the largest instances may be due to the iteration limit for the Bundle algorithm being uniform: possibly, on larger instances more iterations are needed to compute primal solutions of the appropriate quality. Indeed, h-B being faster than h-C precisely in these instances illustrates the nontrivial trade-off between solution time and solution quality.

Next, in Table 4, we asses the performance of our (fastest) heuristic in terms of solution "quality", i.e., comparing the value of the feasible solution it finds against: (*i*) the solution constructed manually, out of experience, by the expert bus planners at the service providers, and (*ii*) the best solution found by `Cplex` directly ran on the full MILP formulation (1)–(6) in "comparable" time. More precisely, we define the "BTS" (best time solver) for each instance as the *fastest* (hence, not necessarily more accurate) between h-B and h-C. Then, with "BT" (best time) the corresponding total running time (ranging from a few minutes to 6 hours), we set the time limit for `Cplex` to BT ("`Cplex*1`"), 2·BT ("`Cplex*2`") and 4·BT ("`Cplex*4`"). An entry "–" means that `Cplex` was not able to find any feasible solution within the time limit.

		ITTVS sol. $\phi_{\%}$: *vs.* BTS			
instance	BTS	Manual	`Cplex*1`	`Cplex*2`	`Cplex*4`
`2Cap_R1`	h-C	117.93	–	–	4.60
`2Cap_R2`	h-C	28.84	1863.83	0.75	0.75
`2Cap_R3`	h-C	35.35	1488.69	1488.69	0.71
`2Cap_F1`	h-C	202.99	3050.46	2.06	**-1.50**
`2Cap_F2`	h-C	51.78	1102.97	885.69	25.72
`2Cap_F3`	h-C	158.86	5338.87	908.09	**-16.23**
`2Cap_F4`	h-C	19.04	–	**-7.88**	**-7.88**
`3Cap_F5`	h-C	252.98	–	48.84	48.61
`3Cap_F6`	h-C	81.38	–	19.61	19.61
`3Cap_F7`	h-B	63.53	–	–	–
`3Cap_M1`	h-B	16.32	–	–	6.11
`4Cap_F8`	h-B	61.55	–	–	–

Table 4: Manual and `Cplex` solutions *vs.* best performing heuristic.

The table shows that our heuristics find much better solutions than the manually obtained ones. The actual value is influenced by the choice of the scaling constant α, and therefore it is not trivial to assess how much better the solutions actually are. However, thanks to the valuable support from experts in *MAIOR*, who regularly work with customers on these issues and therefore have a deep knowledge of the planners' preferences and objectives, it was possible to have it confirmed that the solutions produced by `TTD` are indeed significantly better than the manual ones. We also analyzed how each component (VS and TT) contributes to the improvement with respect to the manual solutions. In general, the VS cost (that represent the service provider true monetary cost) was lower, from slightly to significantly so. This was partly due to saving up to three vehicles, which already is quite significant since vehicles fixed costs are "high" and the total number of required vehicles was between 4 and 24. However, somewhat surprisingly this was not the main component in the savings; most of it was due to a reduction in the waiting times at the terminals. This is actually partly a consequence of the reduction in vehicles (the less the vehicles used, the less the waiting time), but in one case the number of vehicles actually increased (from 3 to 4), and yet the VS cost significantly decreased. On 3 instances out of 12, more than 80% of the improvement was due to the VS cost. On 4 other instances the two components were more balanced, with a VS improvement ranging from 40% to 60%. On the remaining instances, the TT component contributed to more than 70% of the total improvement.

For smaller instances, and allowing much longer times, `CPLEX` sometimes finds better solutions (highlighted in bold), but in general the heuristic approach is quite competitive, especially considering the many cases in which `Cplex` could not find any feasible solution within the time limit. In particular, for complex lines, a direct solution of the MILP formulation, even with a state-of-the-art MILP solver like `Cplex`, is never competitive.

7.2 Second case study: vehicle flow control

In a second case study, we analyze the performance of our `TTD` heuristic, when the number of vehicles for each period is fixed on input, as described in Subsection 5.2. We consider the same set of instances used for the first case study (Table 2), the only difference being the additional information on the number of vehicles allowed to pull-in and pull-out in each time period.

	h-C	h-B	h-B *vs.* h-C		
instance	time	time	time $\phi_{\%}$	LB $\phi_{\%}$	UB $\phi_{\%}$
`2Cap_R1`	3	18	505.60	-0.50	0.01
`2Cap_R2`	6	38	485.59	-0.17	0.68
`2Cap_R3`	2	32	1605.99	-0.12	0.06
`2Cap_F1`	10	10	1.28	-0.52	-1.44
`2Cap_F2`	2	1	-38.61	0.00	0.00
`2Cap_F3`	8	10	29.78	-0.55	-0.25
`2Cap_F4`	1	21	1943.34	-0.11	0.08
`3Cap_F5`	239	305	27.44	-4.11	1.41
`3Cap_F6`	92	153	66.49	-0.34	0.14
`3Cap_F7`	2354	349	-85.18	-2.21	-1.18
`3Cap_M1`	23	57	142.39	-0.27	0.05
`4Cap_F8`	5592	1044	-81.33	-0.60	0.02

Table 5: h-B *vs.* h-C, with vehicle flow control

In Table 5 we compare the performance of the two heuristics in the same way as in Table 3. Roughly, the same trends apparent in the previous results also show off here, i.e., h-B is generally slower except on some complex lines, Lagrangian lower bounds are rather weaker, but h-B solutions are generally better. Yet, the different performance is considerably less marked, perhaps indicating that vehicle flow control constrains much more tightly the set of feasible solutions to the problem, preventing the heuristics to construct much better (or much worse) solutions to one another. Then, in Table 6, we asses the performance of our heuristic in terms of solution quality comparing with manual and `Cplex` solutions, as in Table 4

		ITTVS sol. $\phi_{\%}$: *vs.* BTS			
instance	BTS	Manual	`Cplex*1`	`Cplex*2`	`Cplex*4`
`2Cap_R1`	h-C	5.84	–	0.00	0.00
`2Cap_R2`	h-C	5.63	–	–	**-0.03**
`2Cap_R3`	h-C	2.40	–	–	–
`2Cap_F1`	h-C	8.61	–	**-2.81**	**-2.83**
`2Cap_F2`	h-B	6.53	–	–	0.00
`2Cap_F3`	h-C	9.39	–	**-0.36**	**-0.36**
`2Cap_F4`	h-C	2.59	–	**-0.14**	**-0.14**
`3Cap_F5`	h-C	7.79	–	–	**-2.13**
`3Cap_F6`	h-C	2.53	–	–	50.34
`3Cap_F7`	h-B	9.03	–	–	–
`3Cap_M1`	h-C	4.92	–	–	–
`4Cap_F8`	h-B	4.38	–	–	–

Table 6: Manual and `Cplex` solutions *vs.* best performing heuristic.

Again, with flow control the difference in solution quality is far less pronounced. The heuristics provide better solutions than the manual approach, to the tune of 2-10%; the gain is not very large, but consistent throughout the test set, indicating that the approach can reliably relieve planners with tedious and repetitive handwork. Even when, given much longer running time, `Cplex` can find better solutions than the heuristics, the gain is rather small; again, `Cplex` often fails to find solutions at all, especially for complex lines. Thus, these results confirm that whenever the complexity of the problem increases, a heuristic approach is crucial as even state-of-the-art MILP solvers are not competitive.

8 Conclusions

We have presented a new model for integrated timetabling and vehicle scheduling which, given a set of possible trips and the desired headways, produces a timetable and a set of vehicle schedules with the best (user-defined) compromise between service quality (deviation from the ideal frequency of service) and service cost (number and "cost" of vehicles used). This is a problem that service planners in real-world public transportation companies face day-to-day, and that is usually manually solved through labor and experience.

Our model is based on combining compatibility graphs representations of both the TT subproblems (minimizing deviation from the ideal frequency of service) and the VS subproblem (finding a minimal-cost vehicle schedule), although for the latter a hybrid compatibility/time-space graph formulation is usually preferable. The model is quite flexible and can handle further requirements suggested by *MAIOR* customers, namely complex lines and vehicle flow control. The corresponding MILP formulation is a large-scale multicommodity-type integer program, which is hard to solve in short time with standard techniques. We have therefore proposed a matheuristic approach, that at each iteration solves the continuous relaxation of the problem, either via a general-purpose LP solver or via Lagrangian techniques, and uses the resulting information to drive a diving heuristic.

The approach was tested on real-world instances of the service providers for three major Italian cities; the results show that the heuristic consistently and reliably improves on the solutions obtained manually by experts, thereby indicating that the model can be used to aid even experienced planners in either obtaining better solutions, or obtaining them faster and with less effort, or both. Also, the heuristic is quite competitive w.r.t. the direct use of state-of-the-art, general-purpose solvers like `Cplex`.

The proposed approach lends itself to different uses in an actual operating environment. Most *MAIOR* customers that have tested it use it in a single-line setting like the one envisioned in Section 7. This makes sense in particular for high-intensity lines, possibly using dedicated vehicles (double-length buses, metros, trams, . . .), whereby interlining is necessarily either absent or very reduced. However, the approach can also be used in the standard sequential planning process only to construct the timetables, with the knowledge that a "good" underlining VS then exists for each single (direction of each) line, which can then be improved by a global, inter-line VS step. Indeed, the vehicle schedules produced by the approach can be passed in input to the inter-line VS solver (which is often based on column generation), ensuring that the quality of the VS for the lines where the approach has been used can only improve, exploiting interlining, w.r.t. what is possible considering each line separately.

Actually, it would conceivably be possible to run the model for the whole of a city planning, i.e., for all the lines at the same time: the corresponding problem would however be huge, and making this feasible from the computational viewpoint requires further research.

Also, it could be possible to incorporate in the model other features suggested by *MAIOR* customers, e.g. ones where specific constraints are imposed on the vehicle schedules to ease the construction of feasible crew duties in the last planning phase. Of course, this is always possible by integrating a full-blown constrained VS model in the ITTVS, but this comes at a potentially high computational cost. Alternatives might be possible where the required structures are instead modeled by clever manipulations of the graphs structure, akin to those already illustrated in the paper.

Acknowledgments

The second and third authors gratefully acknowledge the partial financial support from the University of Pisa, project "Modelli ed algoritmi innovativi per problemi strutturati e sparsi di grandi dimensioni" (grant PRA_2017_05), and the Italian Ministry of Education, University and Research (MIUR), project

PRIN 2015B5F27W "Nonlinear and Combinatorial Aspects of Complex Networks". The authors are grateful to Alessandro Bertolini, MSc of the University of Pisa, for his help with coding an early version of the approach. Finally, the authors are very thankful to the bus service operators, among which *ATM* Milan, for providing the instances used for the case studies.

References

[1] Regional bus operation bi-level programming model integrating timetabling and vehicle scheduling. *Systems Engineering Theory & Practice*, 27(11):135–141, 2007.

[2] An integrated approach for timetabling and vehicle scheduling problems to analyze the trade-off between level of service and operating costs of transit networks. *Transportation Research Part B: Methodological*, 70:35–46, 2014.

[3] A.A. Bertossi, P. Carraresi, and G. Gallo. On some matching problems arising in vehicle scheduling models. *Networks*, 17:271–281, 1987.

[4] A. Borghetti, A. Frangioni, F. Lacalandra, and C.A. Nucci. Lagrangian Heuristics Based on Disaggregated Bundle Methods for Hydrothermal Unit Commitment. *IEEE Transactions on Power Systems*, 18(1):313–323, February 2003.

[5] S. Bunte and N. Kliewer. An overview on vehicle scheduling models. *Public Transport*, 1(4):299–317, 2009.

[6] G. Carpaneto, M. Dell'Amico, M. Fischetti, and P. Toth. A branch and bound algorithm for the multiple depot vehicle scheduling problem. *Networks*, 19(5):531–548, 1989.

[7] A. Ceder. *Efficient Timetabling and Vehicle Scheduling for Public Transport*, pages 37–52. Springer, 2001.

[8] A. Ceder and N. Wilson. Bus Network Design. *Transportation Research B*, 20(4):331–344, 1986.

[9] P. Chakroborty, K. Deb, and RK. Sharma. Optimal fleet size distribution and scheduling of transit systems using genetic algorithms. *Transportation Planning and Technology*, 24(3):209–226, 2001.

[10] The Coin-OR Linear Programming project, https://projects.coin-or.org/clp.

[11] G. Desaulniers and M. Hickman. Public Transit. In C. Barnhart G. Laporte, editor, *Transportation*, volume 14 of *Handbooks in Operations Research and Management Science*, pages 69–127. 2007.

[12] J. Desrosiers, Y. Dumas, MM. Solomon, and F. Soumis. Time constrained routing and scheduling. In M.O. Bal, T.L. Magnanti, C.L. Monma, and G.L. Nemhauser, editors, *Trends in Applied Intelligent Systems*, volume 8 of *Operations Research and Management Science*, pages 21–30. 1995.

[13] C. Fleurent, R. Lessard, and L. Seguin. Transit timetable synchronization: Evaluation and optimization. *Technical Report GIRO*, 2007.

[14] C. Fleurent, R. Lessard, and L. Seguin. Integrated Timetabling and Vehicle Scheduling. *Technical Report GIRO*, 2009.

[15] J.P. Fonseca, E. van der Hurk, R. Roberti, and A. Larsen. A matheuristic for transfer synchronization through integrated timetabling and vehicle scheduling. *Transportation Research Part B: Methodological*, 109:128–149, 2018.

[16] A. Frangioni. Generalized Bundle Methods. *SIAM Journal on Optimization*, 13(1):117–156, 2002.

[17] A. Frangioni. About Lagrangian Methods in Integer Optimization. *Annals of Operations Research*, 139(1):163–193, 2005.

[18] A. Frangioni and E. Gorgone. Generalized Bundle Methods for Sum-Functions with "Easy" Components: Applications to Multicommodity Network Design. *Mathematical Programming*, 145(1):133–161, 2014.

[19] A. Frangioni, A. Lodi, and G. Rinaldi. New approaches for optimizing over the semimetric polytope. *Mathematical Programming*, 104(2-3):375–388, 2005.

[20] V. Guihaire and J.-K. Hao. Transit network design and scheduling: A global review. *Transportation Research Part A: Policy and Practice*, 42(10):1251–1273, 2008.

[21] V. Guihaire and J.K. Hao. Improving timetable quality in scheduled transit networks. In N. Garcia-Pedrajas, F. Herrera, C. Fyfe, JM. Benítez, and M. Ali, editors, *Trends in Applied Intelligent Systems*, volume 6096 of *Lecture notes on computer science*, pages 21–30. 2007.

[22] V. Guihaire and J.K. Hao. Transit network timetabling and vehcile assignment for regulating authorities. *Computers and Industrial Engineering*, 59(1):16–23, 2010.

[23] N. Kliever, T. Mellouli, and L.A. Shul. A time-space network based exact optimization model for multi-depot bus scheduling. *European Journal of Operational Research*, 175(1):1616–1627, 2005.

[24] T. Liu and A. Ceder. Integrated public transport timetable synchronization and vehicle scheduling with demand assignment: A bi-objective bi-level model using deficit function approach. *Transportation Research Procedia*, 23:341 – 361, 2017.

[25] T. Liu, A. Ceder, and S. Chowdhury. Integrated public transport timetable synchronization with vehicle scheduling. *Transportmetrica A: Transport Science*, 13(10):932–954, 2017.

[26] The `mcfclass` project, http://www.di.unipi.it/optimize/software/mcf.html.

[27] M. Michaelis and A. Schöbel. Integrating line planning, timetabling, and vehicle scheduling: a customer-oriented heuristic. *Public Transport*, 1(1):211–232, 2009.

[28] H. Petersen, A. Larsen, O.B.G. Madsen, and S. Ropke B. Petersen. The simultaneous vehicle scheduling and passenger service problem. *Transportation Science*, 47(4):603–616, 2012.

[29] V. Schmid and J.F. Ehmke. Integrated timetabling and vehicle scheduling with balanced departure times. *OR Spectrum*, 37(4):903–928, 2015.

[30] P. Serafini and W. Ukovich. A mathematical model for periodic scheduling problems. *SIAM Journal on Discrete Mathemathics*, 2(4):550–581, 1989.

[31] A.P.R Van den Heuvel, J.M. Van den Akker, and M.E. Van Kooten Niekerk. Integrating timetabling and vehicle scheduling in public bus transportation. *Technical Report UU–CS–2008–003*, 2008.

[32] M. Weiszer, G. Fedorko, and Z. Čujan. Multiobjective evolutionary algorithm for integrated timetable optimization with vehicle scheduling aspects. In *Perner's Contacts*, volume 5, pages 286–294, 01 2010.