Branching Strategies to Improve Regularity of Crew Schedules in Ex-Urban Public Transit

Ingmar Steinzen[1], Leena Suhl[2], and Natalia Kliewer[2]

[1] International Graduate School of Dynamic Intelligent Systems, University of Paderborn, Warburger Str. 100, D-33100 Paderborn, Germany
[2] Decision Support & OR Lab, University of Paderborn, Warburger Str. 100, D-33100 Paderborn, Germany,
suhl@uni-paderborn.de,
WWW home page: http://dsor.de

Abstract. We discuss timetables in ex-urban bus traffic that consist of many trips serviced every day together with some exceptions that do not repeat daily. Traditional optimization methods for vehicle and crew scheduling in such cases usually produce schedules that contain irregularities which are not desirable especially from the point of view of the bus drivers. We propose a solution method which improves regularity while partially integrating the vehicle and crew scheduling problems. The approach includes two phases: first we solve the LP relaxation of a set partitioning formulation, using column generation together with Lagrangean relaxation techniques. In a second phase we generate integer solutions using a new combination of local branching and various versions of follow-on branching. Numerical tests with artificial and real instances show that regularity can be improved significantly with no or just a minor increase of costs.

1 Introduction

We discuss timetables in ex-urban bus traffic that consist of many trips serviced every day together with some exceptions that do not repeat daily. In particular, service trips to schools, production facilities, or public swimming baths are often subject to change, e.g., trips may be operated every day except on Sunday, or on Monday only. Unless specifically imposed, traditional vehicle and crew scheduling usually produces *irregular* crew schedules which are undesired in practice. A crew schedule is called irregular if it cannot be repeated many times. Similar to airline crew scheduling (see [10]), *regularity* is an important aspect for crew schedules in public transport since regular solutions can improve operational reliability and can reduce training costs. Furthermore, regular solutions are less error-prone, and crews often prefer to repeat itineraries. In current practice, companies often try to increase regularity of crew scheduling solutions by one of the following heuristic two-phase procedures:

– *All first - irregular second*: First, the planner solves a crew scheduling problem for a particular period with both regular and irregular trips. In a second step, he or she fixes the subset of crew duties that can be operated over the whole period and reoptimizes all unfixed trips. Notice that the second problem may also contain some regular trips.

– *Regular first - irregular second*: The set of service trips is divided into regular and irregular trips. First, a crew scheduling problem for the set of regular trips is solved while the irregular trips are left for subsequent optimization.

In both cases, the second problem has a sparse schedule and, thus, likely requires extensive deadheading, and even its optimal solution yields high costs. On the other hand, if the second problem contains many trips, the corresponding solution has low cost but low regularity as well.

As stated earlier, we are concerned with the regularity of crew schedules and not with the regularity of vehicle schedules. In fact, vehicles are rather insensitive to the quality of their schedules as opposed to drivers. In order to test our approaches, we will concentrate on scenarios where crew scheduling plays the major role. This holds particularly for ex-urban scenarios as we will see in the following section.

As some authors point out, the crew scheduling problem in public transit is basically a multi-criteria optimization problem with operational cost as a very important optimization criterium but involving several others such as number of line changes, total number of duties, number of duties with only one piece of work, and so on. However, to the best of our knowledge, solution approaches to improve the regularity of crew schedules in public bus transport, simultaneously minimizing costs, have not been described in literature before.

We have developed two basic approaches to cope with irregularities in crew schedules. In this paper we propose a novel combination of local branching and follow-on branching that improves the regularity of crew schedules while cost optimality is maintained. As the second approach, [16] compares four bi-objective metaheuristics that include both cost and regularity as objective functions. The latter approach can be used to get a quick estimate of the solution quality obtained with the first approach.

This paper is organized as follows. In Section 2, we give a problem definition for the ex-urban vehicle and crew scheduling problem with irregular timetables. We discuss other approaches related to public (bus) transport from literature in Section 3 and give a formal model definition in Section 4. In the next section, we describe how local branching and user-defined branching rules can be used to steer the solution method to regular crew scheduling solutions. Finally, we provide computational results on real-world and randomly generated instances in Section 6. The paper is concluded with a short summary (Section 7).

2 Problem Definition

2.1 Basic Process of Vehicle and Crew Scheduling

Starting point of the vehicle and crew scheduling process is a timetable that has been determined based on customer demand. A timetable defines a set of trips that are used to carry passengers. Generally, it is assumed that start and end locations for all trips are fixed as well as their start and end times. Given a set of timetabled trips, the vehicle scheduling problem (VSP) can be stated as follows: find an assignment of trips to vehicles such that

– each trip is assigned exactly once,
– each vehicle performs a feasible sequence of trips,
– each sequence starts and ends at the same depot, and
– asset and operational costs are minimized.

Two trips are said to be *compatible* if they can be covered by the same vehicle. Trips operated in sequence by the same vehicle are linked by *deadheads*. Deadheads are vehicle movements or idle times (or both) without carrying passengers. A vehicle is idle if it stands (idle) at a location other than the depot. A *vehicle block* is a sequence of compatible trips that starts with a *pull-out trip* and ends with a *pull-in trip*. A pull-out trip connects the depot with the start location of the first trip while a pull-in trip moves a vehicle from the end location of the last trip to the depot. A daily schedule (*duty*) for one vehicle can thus include several vehicle blocks. Figure 1 depicts an example of a daily schedule for one vehicle with two blocks.

Crew scheduling plays an important role in the operational planning process since crew costs generally dominate vehicle costs. Instead of assigning trips to vehicles as in the preceding phase, we now assign tasks to crews. A basic assumption is that all crews are equal since individual crew members are not considered.

The crew scheduling problem (CSP) is defined as follows: find a set of *duties* for a given set of *tasks* such that

– each task is covered by a duty that can be performed by a single driver,
– each duty satisfies a wide variety of federal laws, safety regulations, and (collective) in-house agreements, and
– labor costs are minimized.

A task is a sequence of activities (such as performing trips or deadheading) between two consecutive *relief points* and represents an elementary portion of work that can be assigned to a driver. A relief point defines a location and time where a driver may change his vehicle. In traditional crew scheduling, i.e., a vehicle first - crew second approach, relief points subdivide vehicle blocks that were obtained in the preceding phase.

A *piece of work* is a sequence of tasks without a (long) break for which a driver stays with the same vehicle. Consequently, duties are composed of pieces of work separated by breaks. Duties start with a *sign-on* and end with a *sign-off* activity. Typically, there are several *duty types* in practical applications, each with a different rule set. Examples of working rules are minimum/maximum driving time, minimum break length, allowed start and end time, or maximum spread (length) of a duty. Moreover, companies often limit the (minimum/maximum) number or percentage of duties of a particular type. For instance, the percentage of split duties that have two pieces of work - one in the early morning and another in the late afternoon with a long break in the middle - is often restricted. Figure 1 shows the schedule of one crew that consists of two pieces of work. Note that the first two tasks remain unassigned.

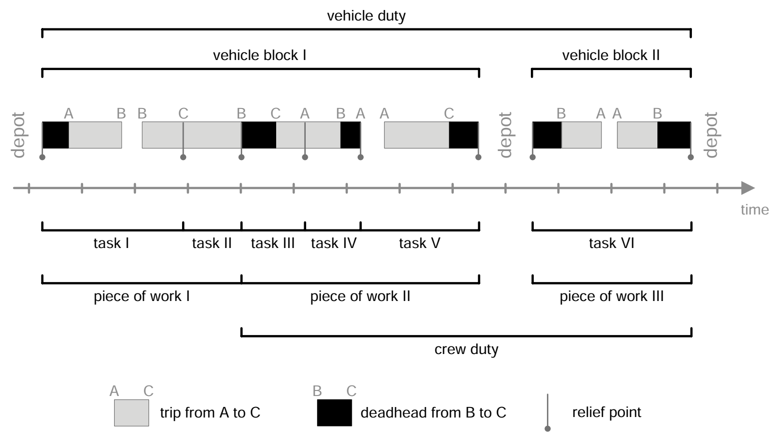

Fig. 1. Schedule of one vehicle and one crew

The objective is often to first minimize the number of duties and second the total working time. Therefore, high fixed crew costs and an hourly rate for working time are taken into account. Crew scheduling problems, however, are often subject to non-linear costs, e.g., overtime bonuses.

[4] shows that the CSP with either working time or spread time constraints is NP-hard. Although duty constraints differ from application to application, we assume that the CSP has at least one of these constraints and is, therefore, NP-hard.

Vehicle and crew scheduling is traditionally approached in a sequential manner which means that vehicle schedules are determined before crew schedules. However, integrating vehicle and crew scheduling and solving both simultaneously can basically reveal further potential to save costs, because of increasing the degrees of freedom and, consequently, size of the solution space.

The *integrated vehicle and crew scheduling problem* (VCSP) for a given set of timetabled trips, depots, and relief points can be stated as follows: find minimum cost sets of vehicle blocks and crew duties such that both vehicle and crew schedule are feasible and mutually *compatible*. Vehicle and crew schedule are compatible if each trip is covered and each deadhead used in the vehicle schedule is also covered by exactly one duty while all deadheads not contained in the vehicle schedule are not part of any duty. The VCSP is NP-hard since (at least) the crew scheduling part is NP-hard.

2.2 Crew Scheduling for Ex-Urban Services

Public transport scenarios can be categorized according to the structure of the underlying transportation network. *Urban* service provides connections within the city while *ex-urban* (*regional*) service connects the city with the suburbs and minor towns in the region of the city. Of course, many companies offer a mixture of both categories. Many regional scenarios have in common that the line network is star-shaped around the depots with only few relief points. Furthermore, distances between relief points are such that drivers are virtually tied to their vehicle in order to reach the relief points. In other words, pieces of work often correspond to vehicle blocks. When traditional vehicle and crew scheduling (vehicles first - crew second) is applied in an ex-urban setting, vehicle blocks are likely to be too long to meet break requirements, or drivers cannot return to their home depot. Conclusively, crews must be scheduled at the same time as vehicles or before vehicles in order to guarantee the feasibility of the crew schedule. In the remainder of this section, we will assume that drivers may only change their vehicles in depots (ex-urban scenario).

Crews can easily be scheduled before vehicles if there is a single depot and vehicle changes outside the depot are not allowed (or drivers can walk from all relief opportunities to the depot). In such a case, we first solve an *independent crew scheduling problem* (ICSP) that we define as follows. Given the traveling times between all pairs of locations and a set of tasks which corresponds to the set of service trips, find a minimum cost set of duties such that all tasks are covered by feasible duties (see also [8]). Since each duty starts and ends at the depot, the vehicle rotations that result from the crew scheduling solution can be put together to form a feasible vehicle schedule (using a vehicle scheduling method). The approach to schedule crews before vehicles is also referred to as *partial integration* (see [1]). However, the number of vehicles is not necessarily minimal in contrast to a fully integrated approach. Notice that a feasible vehicle schedule can also be constructed when there are multiple depots and duties that start and end at the same depot. If continuous attendance is required, and a driver must not stay on his or her (idle) vehicle during a break, each piece of work must start and end at the same depot. As a result, drivers spend their breaks in a depot and take the same or a different vehicle for the consecutive piece of work.

2.3 Vehicle and Crew Scheduling with Irregular Timetables

We will now formally define the vehicle and crew scheduling problem with irregular timetables. Let F be a timetable with tasks $f_1, \dots, f_n$ where task f_i starts earlier than f_{i+1}. Furthermore, a reference crew schedule $R = \{R_1, \dots, R_u\}$ with duties $R_i = \{f_{i1}, \dots, f_{ip}\}$ that is compatible to timetable F is given. The *integrated vehicle and crew scheduling problem with irregular timetables* (VCSP-IT) for timetable $F' \neq F$ and given depots, relief points, and a reference crew schedule R can be stated as follows: find minimum cost sets of vehicle blocks and crew duties such that both vehicle and crew schedule are feasible and mutually compatible. Furthermore, crew schedule $D = \{D_1, \dots, D_v\}$ should have a small *distance* to reference schedule R. A crew schedule with a small distance to reference R is called *similar* or *regular*. However, minimizing costs remains the primary objective.

The perception of distance between two crew schedules can differ from company to company. A very simple *distance measure* is to count the number of duties in the new crew schedule that could not be preserved from the reference crew schedule. In the following, we will describe a more elaborate distance measure that basically counts the number of task sequences not preserved from the reference. Let $Q = F \cap F'$ be the set of *regular* tasks that are part of both timetables. A *regular pair* $S \subseteq Q$ is an ordered pair of regular tasks (f_i, f_{i+k}) that are operated consecutively in both reference R and new crew schedule D. We denote by S^1 the first task of regular pair S while S^2 corresponds to the second task. Notice that an irregular trip may be operated between f_i and f_{i+k}, but no regular trip. Clearly, a regular trip to cannot be at the first (second) position of more than one regular pair. However, it may be at the first position in one pair and at the second in another pair. Furthermore, a *regular chain* $T = (S_1, \dots, S_j) = ((S_1^1, S_1^2), \dots, (S_j^1, S_j^2))$ with $j \geq 1$ and $S_i^2 = S_{i+1}^1, 1 \leq i < j-1$ is an ordered sequence of interconnected regular pairs. $\widetilde{T}$ denotes the number of regular tasks of regular chain T. Furthermore, let $\bar{S}$ and $\bar{T}$ denote the set of all regular pairs and chains, respectively. We define distance measure $\sigma^p(\sigma^c)$ that corresponds to the number of regular tasks that are not part of a regular pair (chain).

$$\sigma^p = |Q| - 2|\bar{S}| \tag{1}$$

$$\sigma^c = |Q| - \sum_{T \in \bar{T}} \widetilde{T} \tag{2}$$

Of course, there are numerous other distance measures possible. However, we believe that our measures give an intuitive approach to regularity of crew schedules. Therefore, we will focus on σ_p and σ_c in the remainder of this paper. However, our approaches also work with other distance measures.

3 Literature Review

In this section, we review state-of-the-art models and solution methods for crew scheduling with irregular timetables from both public transport (bus and railway) and airline perspectives. Since we are concerned about the regularity of crew schedules, we do not consider vehicle scheduling in our literature review. As we will see, the literature on irregular timetables in public bus transport is virtually non-existent. Therefore, we include railway and airline settings in our review.

Solution approaches can mainly be categorized into *regularity* and *rescheduling approaches*. Regularity approaches build a solution from scratch for a given (long) period where the solution should inherently contain as many regular patterns as possible. In rescheduling methods, a reference schedule is given and a new solution for a (short) period is constructed where the new solution should be as similar as possible to the reference. In the following, we will review models and solution methods based on both approaches.

3.1 Regularity Approaches

[18] describe an airline crew scheduling problem with many irregular flights. The authors seek to find a set of pairings (duties) that cover all flights in the planning period (one month) where essentially the total number of *man-days* is minimized. The number of man-days of a pairing is equal to the number of days it lasts. The secondary objective is to minimize costs. Furthermore, a large portion (between 9% and 54%) of all flights is not flown on every day of the planning period. The authors propose a heuristic that systematically merges irregular flights into pairings that only consist of regular flights. Their computational tests involve two real-world data instances with 8,876 and 9,504 flights where the ratio of irregular flights was 54% and 9%, respectively. Their experiments revealed that the instances could be solved in 41 and 92 minutes on an IBM RS/6000 model 900. Moreover, their method could find better solutions than manual planning by experienced engineers. Although the primary objective was to minimize the number of man-days, the approach manages to produce regular crew schedules. For the first instance, 81% of the pairings were regular while 92% of the pairings were flown every day for the second one. However, the authors do not report the impact on operational costs since regular pairings may contain a lot of (paid) waiting time.

[10] introduce the weekly airline crew scheduling model with regularity. The model captures the trade-off between regularity and costs in a weekly schedule. The set of flights is partitioned into groups in such a way that regularity is easily obtainable in each group. A g-regular group for $g = 4, \dots, 7$ contains flights that can be repeated on g consecutive days of the week. By definition, regular flights i from a g-regular group have $g_i \geq g$. Each g-regular group is subsequently partitioned by g-regular pairings. All flights not assigned to a g-regular group, $g = 4, \dots, 7$, are called irregular flights and must be assigned to irregular pairings. In their model, the authors assign penalty costs to irregular flights. Penalty costs decrease with increasing regularity. However, the complete regularity model is intractable and, thus, the authors resort to an approximate model and solution methodology. In particular, pairings are produced in decreasing order of regularity. 7-regular pairings are produced first and an appropriate subset is computed to form 7-regular pairings in the final weekly solution. The flight schedule is reduced by all flights already covered by 7-regular pairings. In the next stage, the remaining flights can only be covered by 6-regular pairings. The process iterates until irregular pairings are generated and the complete flight schedule is partitioned. Computational results with three real-world data instances show that problems with at most 492 flights can be solved in 47 hours computational time. The tests were performed on two clusters: one consisting of

16 machines each with Quad Pentium Pro 200MHz/256 MB main memory and the other comprised of 48 machines each with Dual Pentium II 300MHz/512 MB main memory. The solutions reported improve existing solutions used by the airline both in terms of regularity and costs.

3.2 Rescheduling Approaches

We distinguish between *unplanned* and *planned* rescheduling. Unplanned rescheduling of crews is necessary when the planned crew schedule cannot be executed due to irregular operations or disruptions. Planners usually aim to determine new crew assignments that make as few changes to the original schedule as possible. In other words, planners like to find a new solution with a small distance to the original (reference) solution. Unplanned crew rescheduling is also referred to as crew recovery. Typically, the underlying flight schedule may be changed in crew recovery problems, i.e., flights may be delayed or even canceled, if no feasible recovery scheme is found in a given timeframe. Note that the underlying timetable must not be altered in the problem stated in the preceding section. Furthermore, typical scenarios for crew recovery include local disruptions while irregular trips are often spread over the complete timetable. In conclusion, solution approaches for crew recovery do not seem to be well suited for our problem stated in Section 2. However, recent approaches to airline crew rescheduling (recovery) include, among others, [12], [6], [14], and [13].

In planned crew rescheduling the changes in the underlying timetable are typically known in advance. [9] describes the planned crew rescheduling problem in a railway setting at NS which is the largest passenger railway operator in the Netherlands. At NS crew scheduling is performed in two stages. First, solutions for an annual plan are constructed, i.e., for a general Monday, Tuesday, and so on. In a second phase, the general days are adapted to individual days where specific changes in the timetable for those days are considered. The author states that the changes in the timetable are mainly due to track maintenance or extra service trips that are both usually known in advance. He suggests a set covering formulation where original duties are replaced by new (similar) duties such that all tasks of the modified timetable are covered and total costs of the new duties are minimized. He uses a heuristic based on column generation in combination with Lagrangian relaxation and an elaborate set covering heuristic to compute integer solutions. The computational experiments involve two real-world scenarios and were performed on personal computer with a Pentium IV 3.0 GHz processor/512 MB main memory. The instances with 5,683 and 7,740 tasks had 355 (6.2%) and 827 (10.6%) expired tasks, respectively. For the first instance, only 12.6% of the original duties needed modifications while the ratio increased to 29.5% for the second instance. The author could solve the first instance in approximately 9 hours and the second one in less than 16 hours.

The only approach for public bus transport we are aware of is described in [2]. However, the authors do not provide any details on their approach which is part of the commercial software package HASTUS/CrewOpt (see [5]). They rather emphasize the practical importance of generating efficient solutions that are similar to a reference crew schedule (when the underlying timetable is changed).

4 Mathematical Formulation

In this section, we will give the formulation that will be used in the remainder of this chapter. Recall that we assumed that drivers may only change their vehicles in depots (ex-urban scenario). Therefore, we propose to solve the independent crew scheduling problem (ICSP - see Section 2) first and, then, put the vehicle rotations from the crew scheduling solution together such that the vehicle schedule is feasible. In Section 5 we will seek to improve the regularity of crew schedules for the independent crew scheduling problem.

Let $\mathcal{T}$ be the set of tasks. Furthermore, we define K as the set of all feasible duties and $K(t), t \in \mathcal{T}$ as the set of duties that cover task t. The cost of duty $k \in K$ is denoted by c_k. Finally, decision variables x_k indicate whether duty k is selected in the solution or not. The ICSP can be formulated as set partitioning problem:

$$\sum_{k \in K} c_k x_k \rightarrow \min \tag{3}$$

$$s.t. \quad \sum_{k \in K(t)} x_k = 1 \qquad \forall t \in \mathcal{T}, \tag{4}$$

$$x_k \in \{0,1\}. \tag{5}$$

The objective (3) is to minimize the total costs of the selected duties, and constraints (4) assure that each task will be covered by exactly one duty. When the equality sign in constraints (4) is replaced by a greater or equal sign "$\geq$", we obtain a set covering formulation. Then, tasks may be assigned to more than one driver where the additional drivers are passengers. The set covering formulation is computationally more attractive than the set partitioning formulation (see [20]). In the remainder of this paper, we will consider a set covering formulation.

5 Solution Approaches

5.1 Basic Approach and Test Instances

The purpose of this section is to present two solution approaches that improve the regularity of crew schedules compared to traditional crew scheduling. For both approaches we use model (3)-(5) and apply a column generation algorithm in combination with Lagrangian relaxation. We solve the corresponding Lagrangian dual with a subgradient algorithm to obtain approximate dual values. The column generation pricing problem corresponds to a resource constrained shortest path problem and is solved with a dynamic programming algorithm. For details, see [16] and [11].

The columns generated in the column generation phase serve as input to the second phase where an appropriate integer solution is sought. In the following, we suggest two methods for the second phase that take the trade-off between costs and regularity into account. In particular, we propose a novel combination of local branching and follow-on branching in Section 5.

Our solution approach is based on the observation that (independent) crew scheduling problems have thousands of optimal solutions. This is mainly due to degeneracy.

In Table 1 we give the average number of optimal solutions for independent crew scheduling problems with 80, 100, and 160 trips (tasks). We used the randomly generated test instances from [7]. In accordance with [8] we consider five different types of duties: one tripper type with one piece of work between 30 minutes and 5 hours, and four types consisting of two pieces of work. Each group of a given number of trips involved 10 instances.

We enumerated at most 2,500 different optimal solutions per instance with the branch-and-bound implementation of ILOG CPLEX 9.1.3. The root node of the branch-and-bound tree was solved with a column generation algorithm, i.e. we did not regenerate columns during tree search. As we can see in Table 1, the average number of different optimal solutions can be very high in independent crew scheduling problems. Furthermore, the number of optimal solutions increases if a mere 0.01% deviation to the optimal solution value is allowed.

#trips	#instances	opt. tolerance	
	solved	0.00%	0.01%
80	10	1,052	1,115
100	9	723	945
160	9	1,807	2,046

Table 1. Average number of optimal solutions on Huisman data instances

The basic idea of our solution method is to systematically search an optimal solution among all optimal solutions that is as similar as possible to a given reference solution. In particular, we use *local branching cuts* to select suitable solution subspaces and explore these subspaces with an adapted version of *follow-on branching*. Some preliminary results were presented in [17].

5.2 Local Branching to Find Regular Crew Schedules

Local branching (see [3]) is an exact solution method for general mixed integer programs. The basic idea of local branching is to define suitable solution subspaces that are efficiently explored with a generic MIP solver. In other words, *local branching cuts* are added to *strategically* define subspaces that are *tactically* explored with a black-box solver. The procedure can be viewed as a two-level branching scheme that aims at finding good incumbent solutions at early stages of the computation. The underlying assumption is that small instances of a problem can be efficiently solved with a generic solver while large instances cannot.

Given a feasible *start solution* $\bar{x} \in \{0,1\}^{|K|}$ of ICSP we define the Hamming distance

$$\Delta(x,\bar{x}) = \sum_{k \in L_0} (1 - x_k) + \sum_{k \in K \setminus L_0} x_k \tag{6}$$

where $L_0 = \{k \in K : \bar{x}_k = 1\}$ denotes the *support* of $\bar{x}$. The distance $\Delta(x,\bar{x})$ counts the number of variables in x that flip their values with respect to $\bar{x}$ (either from 1 to 0 or from 0 to 1). For a given neighborhood parameter $\kappa \in \mathbb{N}^+$, the solution space can be partitioned with local branching cuts:

$$\Delta(x,\bar{x}) \leq \kappa \qquad \text{(left branch)}, \tag{7}$$

$$\Delta(x,\bar{x}) \geq \kappa + 1 \qquad \text{(right branch)}. \tag{8}$$

For an appropriate value κ, subspace $\Delta(x,\bar{x}) \leq \kappa$ can be efficiently explored with a generic MIP solver. If the subspace contains a new incumbent $\bar{x}^2$, the scheme is reapplied to the right branch where two new subspaces are constructed: $\Delta(x,\bar{x}^2) \leq \kappa$ and $\Delta(x,\bar{x}^2) \geq \kappa + 1$. On the other hand, if subspace $\Delta(x,\bar{x}) \leq \kappa$ does not contain a new incumbent, the remaining (large) subspace $\Delta(x,\bar{x}) \geq \kappa + 1$ has to be explored with a MIP solver.

For independent crew scheduling, we use a local branching scheme to first explore regions of the solution space that contain solutions similar to a given reference crew schedule R. Similar to equation (1) let σ_k^p be the number of tasks of duty k that are not part of a regular pair. Then, we solve the ICSP (possibly to optimality) with a modified objective function to obtain a start solution $\bar{x}$ as a basis for local branching. The start solution should be similar to the reference crew schedule and should have sufficiently low costs. Therefore, we replace the original cost c_k of column k by $\hat{c}_k = c_k + \alpha\sigma_k^p$ and define α in such a way that σ_k^p dominates the modified cost. Finally, we restore the objective function and use $\bar{x}$ to define the initial neighborhood for local branching.

According to our experience the choice of parameter α is crucial for the performance of the solution procedure. If α is too small, we get a start solution with low costs and low similarity. As a consequence, it is difficult to improve the similarity with local branching. On the other hand, if α is too large, the computational burden to find a minimum cost solution can be very high. In our computational experiments we found that $\alpha \in [150, 400]$ is a robust parameter setting.

5.3 Follow-On Branching to Find Regular Crew Schedules

In order to simplify the exposition, we will briefly recall the basic idea of follow-on branching. Branching on follow-ons relies on a general branching strategy for set partitioning problems that was introduced by [15]. The branching scheme is based on the following property. Given a fractional solution to a set partitioning problem, we can identify two rows (tasks) $f_i \in \mathcal{T}$ and $f_j \in \mathcal{T}$ such that the subset $K(f_i, f_j)$ of columns that contain f_i and f_j has the property

$$0 < \sum_{k \in K(f_i, f_j)} x_k < 1. \tag{9}$$

The remaining fraction of cover for each constraint must be provided by columns that do cover both rows at the same time. Thus, an effective constraint branching scheme is to require to cover two rows f_i and f_j by the same column on one branch and by different columns on the other. [19] slightly modify the scheme to maintain tractability. They only consider trips (rows) f_i and f_j that correspond to trips operated consecutively in a duty (column). Furthermore, the authors show that this modification still constitutes a correct branching scheme. We refer to this strategy as *branching on follow-ons* since we impose which task can follow task f_i in the solution. Moreover, we refer to the task pair (f_i, f_j) as *follow-on.* Notice that each regular pair $S_i \in \bar{S}$ is also a follow-on. In the following, we will describe how follow-on branching is used to construct regular crew schedules.

A regular crew schedule contains as many regular pairs and chains as possible. We modify the follow-on branching scheme in such a way that an (cost) optimal solution has a high regularity as well. In the following, we will propose three novel adaptations of follow-on branching: branching on regular pairs (*fo-r1*), regular chains (*fo-r2*), and pieces of work (*fo-r3*).

The support of a regular pair $(f_i, f_j) \in \bar{S}$ is defined as:

$$g(f_i, f_j) = \sum_{k \in K(f_i, f_j)} x_k. \tag{10}$$

Since we aim at generating regular crew schedules we branch on a candidate regular pair $(f_i, f_j) \in \bar{S}$ where $0 < g(f_i, f_j) < 1$ is satisfied. Branching scheme *fo-r1* selects the regular pair with the best support among all regular pairs.

$$\textit{fo-r1}: \ (f_i, f_j) = \arg \max_{(f_i, f_j) \in \bar{S}} g(f_i, f_j) \tag{11}$$

However, if $\bar{S} = \emptyset$ we choose the follow-on with $f_i, f_j \in \mathcal{T}$ and $\max g(f_i, f_j)$.

Branching scheme *fo-r2* does not rely on the support of single regular pairs, but tries to fix regular chains of maximum length. Recall that $\bar{T}$ is associated with the set of regular chains. Furthermore, we associate $K(T_i)$ with the set of duties that cover regular chain T_i. The set of candidate regular chains $\bar{T}_c$ contains all regular chains $T_i \in \bar{T}$ where $0 < g(T_i) < 1$ with $g(T_i) = \sum_{k \in K(T_i)} x_k$ is satisfied. Algorithm 1 depicts branching scheme *fo-r2* where we try to branch on a regular chain of maximum length if there are candidate chains.

Notice that scheme *fo-r2* corresponds to the latter scheme *fo-r1* if the set of candidate regular chains $\bar{T}_c$ only consists of chains of length two.

Algorithm 1: Branching on regular chains (*fo-r2*)

```
Find candidates
Compute set of candidate regular chains $\bar{T}_c = \{T_i : 0 < g(T_i) < 1\}$.
Branching
if $\bar{T}_c \neq \emptyset$ then
    Branch on follow-on $f_i, f_j \in \mathcal{T}$ with $\max g(f_i, f_j)$
end
else
    Initialize $\bar{T}_c^{\max} = \{T_i \in \bar{T}_c : |T_i| = \max_{T_j \in \bar{T}_c} |T_j|\}$
    Branch on regular chain $T_i \in \bar{T}_c^{\max}$ with $\max g(T_i)$
end
```

Finally, we propose branching scheme *fo-r3* where we branch on a piece of work whenever that piece of work forms a regular chain. If several pieces correspond to candidate regular chains, we select the piece with the maximum number of tasks. Algorithm 2 presents how branching on regular pieces of work is performed.

Algorithm 2: Branching on regular pieces of work (*fo-r3*)

```
Find candidates
Compute set of candidate regular chains $\bar{T}_c = \{T_i : 0 < g(T_i) < 1\}$.
Branching
if $\bar{T}_c \neq \emptyset$ then
    Branch on follow-on $f_i, f_j \in \mathcal{T}$ with $\max g(f_i, f_j)$
end
else
    if $\exists T_i \in \bar{T}_c : T_i$ is piece of work then
        Initialize $\bar{T}_{cp} = \{T_i \in \bar{T}_c : T_i$ is piece of work$\}$
        Branch on regular chain $T_i \in \bar{T}_{cp}$ with $|T_i| = \max_{T_j \in \bar{T}_{cp}} |T_j|$ and
        $\max g(T_i)$
    end
    else
        Initialize $\bar{T}_c^{\max} = \{T_i \in \bar{T}_c : |T_i| = \max_{T_j \in \bar{T}_c} |T_j|\}$
        Branch on regular chain $T_i \in \bar{T}_c^{\max}$ with $\max g(T_i)$
    end
end
```

5.4 Local and Follow-On Branching to Find Regular Crew Schedules

Local branching and follow-on branching can be combined. In particular, we embed follow-on schemes *fo-r1* to *fo-r3* into local branching to explore neighborhoods $\Delta(x, \bar{x}) \leq \kappa$. We hope to explore neighborhoods $\Delta(x, \bar{x}) \leq \kappa$ in such a way that (1) an new incumbent is found fast and (2) the new incumbent has a smaller distance than other solutions in the neighborhood. If the reference solution is of high quality, a valuable follow-on might be selected first and might reduce the computational time to explore the neighborhood. To sum up, we *strategically* define subspaces with local branching and *tactically* explore them with follow-on branching.

6 Computational Results

We test our approaches on real-world and randomly generated data instances. We consider two real-world and eight randomly generated data instances. The artificial instances were generated as described in [8]. However, all instances have a single depot and drivers may only change their vehicle in that depot. We make these assumptions in order to reflect a typical ex-urban scenario (see Section 2). Furthermore, we assume that a reference crew schedule is known for each data instance.

In Table 2 we give details on the data instances that result from solving the linear relaxation of the ICSP with a column generation algorithm. The last two instances correspond to real world problems while the others were randomly generated. We report the ratio of irregular trips in percent (*%irr*), the number of rows (*#rows*), columns (*#cols*), and non-zeros (*#nnz*). For each data instance the ratio of irregular trips refers to the number of new trips, i.e., trips that are not in the reference schedule, compared to the total number of trips. In the second part of the table we give details on the column generation phase: the number of iterations (*#iter*), and the computational time spend on master (*cpu_ma*) and pricing problem (*cpu_pr*). To maintain comparability between both approaches, we used operating costs as single objective in the column generation phase. In addition to the assumptions stated above we apply the following parameter settings for our branching approach:

The computational time to find an integer solution is limited to 2 hours (7,200 seconds). In our local branching implementation, at most 20% of the variables of the incumbent may flip their values. Furthermore, the computational time to explore subspaces $\Delta(x, \bar{x}^i) \leq \kappa$ (left branches) is limited to 15 minutes (900 seconds). If the time limit is reached and no new incumbent is found, we reduce the size of the subspace by 50% to speed-up its exploration. For further details we refer to [3].

All computational experiments with the branching schemes were performed on a personal computer running Windows XP with an Intel Pentium IV 2.2 GHz processor and 2 GB of main memory.

instance	%irr	#rows	#cols	#nnz	#iter	cpu_ma	cpu_pr
art320_1	5.0	320	100,944	857,215	31	245	140
art320_2	5.0	320	60,128	384,478	21	143	85
art400_1	5.0	400	72,673	459,906	22	125	122
art400_2	5.0	400	57,769	352,592	21	130	77
art640_1	5.0	640	156,044	1,227,320	41	1,006	1,673
art640_2	5.0	640	104,595	643,113	28	572	695
art800_1	5.0	800	135,572	852,337	37	1,060	2,054
art800_2	5.0	800	162,209	1,158,539	39	1,773	2,887
real430	4.4	430	98,710	1,204,084	31	391	297
real433	4.8	433	103,516	1,236,954	31	411	257

Table 2. Description of data instances

In Table 3 we show results on the regularity of crew schedules when we apply local branching (*locbr*) and follow-on branching (*fo-r1, fo-r2, fo-r3*) as described in Section 5. Furthermore, we compare our method with the default branch-and-bound implementation of ILOG CPLEX 9.1.3 (*cpx-def*) and local branching in combination with default branching of CPLEX (*locbr_cpx-def*). For each method we give the average over the ten instances described in Table 2. In Table 3 we report the computational time in seconds spent in the second (integer) phase (*cpu_ip*), the optimality gap in percent (%gap) and three regularity measures. The regularity measures are defined as follows. The percentage of preserved duties (*%prd*) refers to the percentage of duties in the new crew schedule that could be (exactly) kept from the reference crew schedule. Similarly we define the percentage of preserved regular pairs (*%prp*). The average regular chain length of a crew schedule corresponds to the average number of regular tasks in a duty. In this context, the percentage of the average chain length (*%avgcl*) refers to the average regular chain length of the new crew schedule compared with average regular chain length of the reference crew schedule. For example, if the reference schedule has on the average 8 regular tasks per duty, and the average regular chain length in the new crew schedule is 4 tasks, then $avgcl = \frac{4}{8} = 50\%$.

As can be seen from Table 3 branching scheme *fo-r1* provides the best results in terms of solution time and solution quality. Recall that objective function and, thus, solution quality refer to operational costs. On the other hand, local branching considerably improves the regularity of the new crew schedules, e.g., the number duties that can be kept from the reference. Basically, we generally observe an increase of solution time and decrease of solution quality if local branching is used. However, local branching in combination with scheme *fo-r1* gives a better solution quality than the default version of CPLEX. To sum up, we conclude that local branching effectively improves the regularity while follow-on branching scheme *fo-r1* is well suited to improve solution quality and time. The combination of both methods leads to improved solutions in terms of both cost and regularity compared to a traditional approach with CPLEX. A reason for

			regularity measures		
method	cpu_ip	%gap	%prd	%prp	%avgcl
cpx-def	2,437	1.93	6.3	53.5	31.0
fo-r1	2,095	0.42	7.7	54.4	31.2
fo-r2	3,649	2.20	8.2	56.8	33.7
fo-r3	4,247	2.81	6.6	55.0	32.5
locbr_cpx-def	6,420	2.60	27.4	79.0	50.1
locbr_fo-r1	5,492	1.55	28.0	80.2	51.2
locbr_fo-r2	5,806	3.81	32.3	81.1	54.5
locbr_fo-r3	6,270	3.70	25.6	80.0	51.2

Table 3. Results on regularity for branching approaches

the good performance of *fo-r1* might be that branching on sequences from the reference leads to high quality solutions if the reference schedule is also of high quality.

7 Summary

In this paper, we discussed the ex-urban vehicle and crew scheduling problem with a single depot and irregular timetables. Unless specifically imposed, traditional vehicle and crew scheduling usually produces irregular crew schedules which are undesired in practice. We presented solution approaches that improve the regularity of crew schedules compared to traditional crew scheduling. In particular, we proposed a novel combination of local branching and follow-on branching. A computational study that involved randomly generated and real-life data showed the applicability of the proposed techniques. In fact, our branching scheme lead to improved solutions in terms of both cost and regularity compared to a traditional approach with CPLEX. A current limitation of our approach is that we do not consider a full integration of vehicle and crew scheduling. Instead, we focussed on an ex-urban scenario where drivers are virtually tied to their vehicle.

References

1. R. Borndoerfer, A. Loebel, and S. Weider. A bundle method for integrated multi-depot vehicle and duty scheduling in public transit. Technical Report ZR-04-14, ZIB - Zuse Institute Berlin, Berlin, Germany, 2004.
2. A. Dallaire, C. Fleurent, and J.-M. Rousseau. Dynamic constraint generation in crewopt, a column generation approach for transit crew scheduling. Technical report, GIRO Inc., Montreal, Canada, 2004.
3. M. Fischetti and A. Lodi. Local branching. *Mathematical Programming*, 84:23–47, 2003.

4. M. Fischetti, A. Lodi, S. Martello, and P. Toth. The fixed job schedule problem with working-time constraints. *Operations Research*, 37(3):395–403, 1989.
5. GIRO. Hastus transit scheduling and operations. Available at http://www.giro.ca/en/products/hastus/index.htm, July 2007.
6. Y. Guo, L. Suhl, and M. P. Thiel. Solving the airline crew recovery problem by a genetic algorithm with local improvement. *Operational Research An International Journal*, 5, 2005.
7. D. Huisman. Random data instances for multiple-depot vehicle and crew scheduling. Available at http://www.few.eur.nl/few/people/huisman/instances.htm, April 2005.
8. D. Huisman. *Integrated and Dynamic Vehicle and Crew Scheduling.* PhD thesis, Tinbergen Institute, Erasmus University Rotterdam, 2004.
9. D. Huisman. A column generation approach to solve the crew re-scheduling problem. *European Journal of Operational Research*, 180:163–173, 2007.
10. D. Klabjan, E. Johnson, G. Nemhauser, E. Gelman, and S. Ramaswamy. Airline crew scheduling with regularity. *Transportation Science*, 35:359–374, 2001.
11. N. Kliewer, T. Mellouli, and L. Suhl. A time-space network based exact optimization model for multi-depot bus scheduling. *European Journal of Operational Research*, 175(3):1616–1627, 2006.
12. L. Lettovsky, E. Johnson, and G. Nemhauser. Airline crew recovery. *Transportation Science*, 34:337–348, 2000.
13. C. Medard and N. Sawhney. Airline crew scheduling: From planning to operations. *European Journal of Operational Research*, 183:1013–1027, 2007.
14. R. Nissen and K. Haase. Duty-period-based network model for crew rescheduling in european airlines. *Journal of Scheduling*, 9:255–278, 2006.
15. D. M. Ryan and B. Foster. An integer programming approach to scheduling. In A. Wren, editor, *Computer Scheduling of Public Transport: Urban Passenger Vehicle and Crew Scheduling*, pages 269–280. Amsterdam, North-Holland, 1981.
16. I. Steinzen. *Topics in Integrated Vehicle and Crew Scheduling in Public Transit.* PhD thesis, DSOR Lab, University of Paderborn, 2007.
17. I. Steinzen, V. Gintner, and L. Suhl. Local branching und branching-strategien fuer umlauf- und dienstplanung im regionalverkehr mit unregelmaessigen fahrplaenen. In H.-O. Guenther, D. Mattfeld, and L. Suhl, editors, *Management logistischer Netzwerke: Entscheidungsunterstuetzung, Informationssysteme und OR-Tools*, pages 407–424. Physica-Verlag, Heidelberg, 2007.
18. A. Tajima and S. Misono. Airline crew-scheduling with many irregular flights. In H. Leong, H. Imai, and S. Jain, editors, *Lecture Notes in Computer Science: Proceedings of the 8th International Symposium on Algorithms and Computation - ISAAC97*, pages 2–11. Springer, Heidelberg, 1997.
19. P. H. Vance, A. Atamtuerk, C. Barnhart, F. Gelman, E. Johnson, A. Krishna, D. Mahidhara, and R. Rebello. A heuristic branch-and-price approach for the airline crew pairing problem. Technical Report LEC-97-06, Georgia Institute of Technology, Atlanta, USA, 1997.
20. F. Vanderbeck. *Decomposition and Column Generation for Integer Programs.* PhD thesis, Universite Catholique de Louvain, 1994.