CEJOR (2013) 21 (Suppl 1):S75–S94
DOI 10.1007/s10100-012-0274-3

ORIGINAL PAPER

An efficient solution approach for real-world driver scheduling problems in urban bus transportation

Attila Tóth · Miklós Krész

Published online: 26 October 2012

Abstract The driver scheduling problem in public transportation is defined in the following way. There is a set of operational tasks, and the goal is to define the sequence of these tasks as shifts in such a way that every task must be assigned to a shift without overlapping. In real-world situations several additional constraints need to be considered, which makes large practical problems challenging to be solved efficiently. In practice it is also an important request with respect to a public transportation scheduling system to offer several versions of quasi-optimal solutions. In this paper we present an efficient driver scheduling solution methodology which is flexible in the above sense.

Keywords Optimization for public transportation · Driver scheduling · Decision support · Real-world applications

1 Introduction

For transportation companies, the largest expense is induced by the operational costs. One of the most common ways to reduce the cost is the introduction of a computer supported integrated information system. Such a practical system frequently contains some optimization tool for scheduling, in which driver scheduling plays a central role. On the other hand, because of the high number and complex constraints, driver scheduling is typically one of the most difficult problems in transportation scheduling.

A. Tóth · M. Krész (✉)
Juhász Gyula Faculty of Education, Institute of Applied Sciences,
University of Szeged, Boldogasszony sgt. 6, 6725 Szeged, Hungary
e-mail: kresz@jgypk.u-szeged.hu

A. Tóth
e-mail: attila@jgypk.u-szeged.hu

Since driver scheduling is generally just a sub-step of the global scheduling process in public transportation framework, many times it is a part of a decision support system. In such a system the main claim is the widely changeability in parameters and the modularity in the method. Usually the usage of this kind of real life application is to generate different solutions with different parameter settings and trying different situations and compare them, examine its effect to the other steps of the global scheduling. Because of the above mentioned requirements, there is a growing need in practice for interactive decision support scheduling systems. In such a system, it is expected from a method to provide solutions relatively fast for a whole planning period (typically 1–2 months). In this paper we present a solution approach for optimizing driver schedules, which has the above-mentioned flexibility and efficiency.

The structure of the paper is the following. First we summarize the background of driver scheduling problems. Then our general methodology is described, by introducing the so-called CAJ approach. In the next section, the framework of our new efficient real-world driver scheduling approach is discussed in detail. Finally, a case study will be presented for showing the applicability of our new methodology.

2 Driver scheduling problems in urban transportation

Scheduling problems arising in public transportation are highly challenging tasks. For transportation companies, a global optimum of the operational costs induced by both the vehicle and driver scheduling is expected. Though some combined models solving simultaneously both scheduling tasks have been recently worked out (cf. Steinzen et al. 2010), these methods, because of the size and complexity of the problem, are suitable for small problem instances only. Therefore, in practice, still the classical sequential optimization approach of the scheduling subproblems can be applied.

In this section first we review the above approach of optimization processes, then we present the general problem of driver scheduling supplemented by the real-world constraints and objectives.

2.1 Optimization process

Generally, in real-world situations the optimization process is divided into three subproblems: vehicle scheduling, driver scheduling and driver rostering. Although these tasks have strong influence on each other, and the solution of a phase can restrict the next one, the above decomposition is necessary. In the following, we shortly review the above subproblems.

2.1.1 Vehicle scheduling

Vehicle scheduling problem consists of scheduling a fleet of vehicles to cover a set of tasks at minimum cost. The tasks are given by prescribed time intervals, starting and ending place, and the vehicles are supplied by different depots. The problem is to minimize the total cost induced by the number of vehicles used and by the vehicle travel (timetabled and deadhead trips) length. There are several mathematical models

for this problem. The most widely used ones are when the problem is formulated as an integer multi-commodity network flow model. (cf. Kliewer et al. 2006)

2.1.2 Driver scheduling

Optimization in staff scheduling is nowadays a central question, the most typical applications come from the operations of hospitals (nurse scheduling), call centers (scheduling of operators) and transportation companies (driver scheduling). In staff scheduling, the constraints are highly problem-specific, thus problem statements, models and solution methods are quite various in different fields (Ernst et al. 2004). In driver scheduling, there is a set of operational tasks and the goal is to define the sequence of these tasks as shifts in such a way that every task must be assigned to a shift without overlapping. There are many constraints depending on the transportation company and the relevant laws, such as maximum working hours, length of the daily break, number and length of short breaks. The most favorite approaches are set partitioning or set covering as relaxation of the problem. Because of these differences and the high number of hard constraints, most solution methods deal with only a few basic constraints, such as maximum working hours or handling some short and a long (meal) break. (see eg. Desrochers and Soumis 1989)

2.1.3 Driver rostering

Driver rostering is a member of the general staff rostering problem where the task is to assign the daily shifts to the employees with some general and special constraints. The constraints of driver rostering generally correspond to the constraints of other typical applications such as nurse rostering (Cheng et al. 1997), crew rostering (Cappanera and Gallo 2004; Caprara et al. 1998; Yunes et al. 2005) or operator rostering in call-centers (Segal 1974). One can find an overview for crew rostering in Ernst et al. (2004) and a general model for real-world applications in Nurmi et al. (2011).

2.2 Models and algorithms for driver scheduling

In this subsection we review the main concepts, models and solution methods applied for driver scheduling in the literature. First we summarize the basic notions.

A geographical place where a *task* is started or finished, and a driver can get on or off the vehicle are called *relief points*. If a vehicle meets a relief point, we talk about a *relief opportunity*. The vehicle task between two consecutive relief opportunities is called as a *work-piece*.

There are two general approaches for the driver scheduling problem. The first one is the Generate and Select (GaS) approach, which generates a set of candidate feasible shifts and select a subset fulfilling the rules with minimal cost (Dias et al. 2002; Guerinik and Caneghem 1995; Kwan et al. 2001; Li and Kwan 2005; Steinzen et al. 2009; Wren et al. 2003; Yunes et al. 2005). The other one is the Constructive approach, which defines a feasible solution and usually tries to improve it (Aickelin et al. 2006; Caviquel et al. 1999).

In GaS, the aim of the generating phase is to produce a suitable large number of candidate feasible shifts. Obviously, generating numerous shifts requires a significant amount of time, but only a few shifts restrict the success of the selection phase and the quality of the solution. The complexity of the generating phase mostly depends on the number of the tasks and that of the rules. Nevertheless, the complexity of the rules also strongly influences the hardness of the problem. The selection phase of the GaS traditionally formulated as a set covering or set partitioning problem. With a set of tasks and a set of generated feasible candidate shifts, the formulation is the following:

$$\begin{aligned} &\text{minimize} \quad w_1 \sum_{j=1}^{n} c_j x_j + w_2 \sum_{j=1}^{n} x_j \\ &\text{subject to} \quad \sum_{j=1}^{n} a_{ij} x_j \geq 1, \quad i = 1, 2, \ldots, m \\ &x_j = 0 \quad \text{or} \quad 1, \; j = 1, 2, \ldots, n \end{aligned}$$

where

n = the number of candidate shifts
m = the number of tasks
x_j = shift variable, $x_j = 1$ if shift j is selected and 0 otherwise
c_j = cost of shift j $a_{ij} = 1$ if taks i is covered by shift j, 0 otherwise
w_1 and w_2 are weights

With this formulation the problem is a set covering, where overlapping is allowed, i.e. several drivers can be assigned to one task. In this case, the number of overlaps has to be minimized. The set partitioning model is a special case, when overlapping is not allowed (the inequalities become equalities). However, in this case, the existence of a feasible solution is not guaranteed. There are numerous solution methods to solve such problems, such as integer linear programming (Wren et al. 2003; Yunes et al. 2005), using branching techniques (Steinzen et al. 2009), constraint programming (Guerinik and Caneghem 1995; Yunes et al. 2005), different heuristic technics (Dias et al. 2002; Kwan et al. 2001) or some kind of hybrid model (Yunes et al. 2005).

The Constructive approach builds a solution guided by an optimization method. The classical way of this framework is the Iteration approach which generates an initial solution and iteratively improves it (Aickelin et al. 2006; Caviquel et al. 1999).

The most crucial task is to check the feasibility of solutions— meaning the feasibility of all the shifts—during both the generating construction processes. Since the number of feasible shifts is extremely high even in the case of a small problem, most of the technics make some simplifications in order to decrease the running time. Generally the problem size is reduced by omitting relief opportunities, or just a smaller treatable set of shifts is generated to search the best one. Naturally, these simplifications affect the optimization and limit the success of the method. The other general way of simplification is to omit some rules of higher complexity. Typically the break rules are simplified: special break assignment and use a single mealbreak and/or short break, or work without break at all. Clearly, the above modifications make these methods unsuitable in industrial use. In real-world projects the demand is to handle all the

rules and find at least a quasi-optimal solution in reasonable time. Therefore, none of the above simplifications are acceptable in practice.

2.3 Real-world constraints of driver scheduling

For optimizing driver scheduling, the constraints are really crucial for the solution method: they strongly affect the quality of the solution and the running time. In highly constrained real life problems it is a quite difficult question to find the balance between the above aspects. Since usually the problem size is large and the prescribed rules are fixed, the main aim is to find an efficient method providing a good quality feasible solution from practical point of view. The main constraints are defined on national and international level: apart from governmental rules, there are many regulations decreed by the EU, for example. Furthermore, transportation companies apply local claims as hard or soft constraints.

The most important claim against real life applications is the flexibility. In many cases, the adaptability and the running time are the most important aspects, if the quality of the solution satisfies certain requirements (typically some constraints). In case of decision support for scheduling, certain engineering requirements concerning the quality of the solution cannot be formalized, thus long term planning in scheduling is realized by an interactive mechanism. This flexibility is a central question with respect to the other phases of the optimization process: the evaluation of a solution is possible together with the vehicle scheduling and driver rostering only. Therefore, adaptation mostly depends on the possibility to define rules and set the parameters so that the algorithm is able to produce different solutions.

2.4 Objective

In driver scheduling, the quality of a solution only partially depends on the number of resulted shifts. More generally, the cost of a solution is considered as the sum of the costs of shifts contained in. Naturally, there is a general cost of employing a driver, thus minimizing the total cost of driver scheduling will keep the number of shifts on a low level. On the other hand, costs are assigned to shifts according to the contained tasks. Furthermore, there can be defined some special features—such as shift type, expected length of the shifts, etc.—which can appear in the cost even as a penalty. Nevertheless, in a most general sense, for evaluating a solution, the working time of shifts as total cost can be considered.

3 Methodology

Our scheduling method is divided into two main phases. The first one is the basic algorithm which produces the rough shifts. These *rough shifts* contain only the trips (both timetabled and deadhead trips) and the travelling activities of the driver. The second phase, the completion, produces the complete shifts containing all the obligatory activities and fills the idle times with the proper idle activities. Most of the methods in

the literature deal with the first phase only and do not care about the activities and rules considered in completion. However, practical solutions require valid shifts concerning all the real-world circumstance.

The main idea of the first phase is dividing the driver scheduling into sequential substeps where different optimization methods can be used and combined. Applying this approach, high flexibility can be reached, since quite easy to adjust the process to the expectations. Moreover, there is a possibility to improve the output of a step before the next one manually, as sometimes a human expert can consider constraints which are formally hard to describe.

The completion phase is the local specific part of driver scheduling. In each application it must be suited to the local circumstances. Several local rules and specific obligatory and optional duties can be defined for the complete shifts. This phase fits the shifts to the local demands and handles all the defined activities. The real-world final cost can be calculated after completion only, since the costs of local activities influence the objective value. It is true in the general case, when only the working time is considered as cost measurement, since the working time of duties inserted during the completion phase needs to be added as well. Nevertheless, because of the company-specific constraints of the completion, general methods cannot be worked out for this phase. Consequently, with respect to our new two-phase approach, the basic algorithm needs to fulfill the following requirements:

- *Efficiency* The algorithm needs to be efficient enough to be able to solve large real-world problems in a decision support way: several alternative versions are to be considered.
- *Flexibility* The algorithm needs to offer appropriate parameter settings for the above decision support analysis, and during the method, the opportunity of human interaction should be guaranteed.
- *Two-phase property* Though the completion phase is company-specific, by appropriate parameter settings, the solution produced by the basic algorithm needs to be also feasible and have a good quality after finalizing its rough shifts by the second phase.

In this section, we introduce our new Cut and Join (CAJ) Approach having all the above properties.

3.1 Basic notions and notations

The set of relief points (where activities start or end) is denoted by P. Each relief point can have special attributes (e.g. it can be a place for spending breaks). The set of driving activities is denoted by A, and each activity $a \in A$ has seven properties: a^{st} is the starting time, a^{et} is the ending time, a^{sp} is the starting relief point, a^{ep} is the ending relief point, a^{type} is the type, a^{cost} is the cost of the activity and a^{v} is the assigned bus (for activities not requiring buses, its value is *null*). In the completion phase, there might be several different types of activities, for the basic algorithm, however, we distinguish between three types only: vehicle task activity (practically driving activity), travelling activity, idle time.

Compatibility of two activities means that they can be contained in the same shift; this concept is applied in a natural sense: compatible activities are not overlapped in time, they must be geographical sequential (where a task is finished, the next one must be started) or a new task must be inserted for accomplishing the connection between them (i.e. a travelling of the driver). The other important requirement is defined for the sequence of activities. A *legal sequence* of activities is also a natural concept depending on the rules defined by nationally and internationally (typically governmental and EU rules): they are mostly about regulating the working time, driving time, rest periods (between two shifts when the driver may freely dispose of his time) and breaks (part of the shift when the driver must not carry out any work). Moreover, local transportation companies define many other rules or tighten the previous regulations fitting to the transportation area, the local circumstances and the attitude of the company. Then a *legal shift* is a legal sequence of pairwise compatible activities.

The input for the driver scheduling is given by a set T of vehicle tasks, as vehicle schedules. Similarly to driver activities, a task $t \in T$ is given by seven properties $t = (t^{st}, t^{et}, t^{sp}, t^{ep}, t^{type}, t^{cost}, t^{v})$. A feasible solution of driver scheduling is a set S of legal shifts covering all the vehicle tasks, the cost of which is defined by summing up the costs of the shifts contained in. The objective is to minimize the cost over all feasible solutions. The planning period is usually 1 day, so the optimization works day-by-day and considers only those rules which concern daily duties.

3.2 The Cut and Join (CAJ) Approach

As it was pointed out, most of the activities (besides the driving trips and travelling) are inserted into the shifts during the completion phase only. Nevertheless, the basic algorithm needs to take into consideration the above constraints by appropriate parameter settings. Our new flexible approach named Cut and Join (CAJ) is outlined below; it is described in detail in the next section.

The algorithm goes through the following steps:

1. *Creating work-pieces* This step creates work-pieces by pulling the tasks together by the possibility of driver changing.
2. *Creating pre-shifts* This step generates initial solutions, called pre-shifts, omitting the "difficult" rules.
3. *Cutting* This step slices the pre-shifts into smaller feasible parts considering all the rules.
4. *Joining* In this step the shift parts are joined together producing longer feasible shifts.
5. *Refining* This final step improves the solution by exchanging work-pieces between the shifts.

3.3 Objective function

Since the evaluation of a solution can be calculated after the completion phase only, during the CAJ algorithm, we use an objective function constructed in a way considering the required two-phase property. As mentioned before, during the optimization process, we take into consideration only the vehicle tasks, the driver travelling

and the idle time as activities. By a statistical analysis, a cost is assigned to the gaps between two consecutive tasks. The costs of these idle periods are calculated by the statistics of a *training scheduling*, typically given by the shifts used presently by the considered transportation company. Let S' be the training scheduling and $s' \in S'$ be a shift. An *idle time* $p_n \subset s'$ is defined by two vehicle tasks t_i, t_{i+k-1} in $s' = (\ldots, t_i, a_{i+1}, a_{i+2}, \ldots, a_{i+k-1}, t_{i+k}, \ldots)$ where $t_r \in T$ are vehicle tasks and $a_r \notin T$ are other activities. Than the *category (rank)* of this period is defined by its length:

$$p^r = \left\lfloor \frac{t^{st}_{i+k} - t^{et}_i}{5} \right\rfloor$$

which means a rank in a time scale R of 5 min. The cost of this period is calculated by summing up the cost of the inner activities:

$$p^{\cos t} = \sum_{j=i+1}^{i+k} a_j^{\cos t}.$$

The cost of a time period R_m in the scale is calculated by the average of the costs of these idle periods categorized to this rank.

$$R_m^{\cos t} = \frac{\sum_{j=1}^{|R_m|} p_j^{\cos t}}{|R_m|},$$

where $p^r_j = m$.

Using the above scale, the cost of a shift is calculated as a weighted sum of the vehicle tasks, the traveling activities and the idle periods.

$$c(s) = \alpha * \sum_{t \in T} \left(t^{et} - t^{st}\right) + \beta * \sum_{a \in A\prime} \left(a^{et} - a^{st}\right) + \gamma * \sum_{i=1}^{|s|-1} R_{p_i}$$

where $A' = \{a : a^{type} = \text{TRAVELLING}\}$ is the set of the travelling activities, p_i-s are the idle periods in s and α, β, γ are nonnegative weights.

Since practically the bus lines and trips are correlated with the general and local rules, the best idea is to use the present local bus scheduling to build up this rank. Evidently, it can be updated later manually or by the produced scheduling.

4 Solution method

The input of driver scheduling is the scheduled duties of the vehicles (regular trips, deadhead trips etc.) which driver must be assigned. The aim of the basic algorithm is to schedule these vehicle tasks into driver shifts, considering the corresponding rules. In this section we introduce the idea of each step of the algorithm. Nevertheless, the main value of our methodology is the solution approach itself; the presented algorithmic solutions of each step are used to demonstrate possible efficient realizations.

The methods used throughout the algorithm are mainly based on greedy techniques. This well-known algorithmic approach uses local information in decision situations for choosing the best direction in order to continue the optimization process. In addition to its extremely fast computational performance, the main advantage of the greedy methodology is its easy-to implement feature. Since, in our case, one of the main demands is the fast running time, the greedy approach offered a straightforward way in demonstrating the power of our new solution method. Nevertheless, the flexible framework allows applying more sophisticated techniques as well.

4.1 Creating work-pieces

A driver changing activity on a vehicle needs a prescribed time period, which is important to be considered for assigning drivers to vehicle tasks. On the other hand, there can be relief points where driver changing is forbidden. Because of the above reasons, consecutive vehicle tasks can be involved into one combined driver task, as there is no possibility to change the driver on the vehicle between the trips. Summarizing the above facts, a *work-piece* is defined as one or more consecutive tasks of the same vehicle, which must be carried out by the same driver. The process of creating work-pieces can be executed with a naive method in linear time.

Let WP denote the set of the work-pieces and $wp \in WP$ be a single work-piece. If t_j and t_{j+1} are two consecutive tasks of the same vehicle, then $wp^{st} = t_j^{st}$, $wp^{et} = t_{j+1}^{et}$, $wp^{sp} = t_j^{sp}$, $wp^{ep} = t_{j+1}^{ep}$ and $wp^v = t_j^v = t_{j+1}^v$ iff $t_j^{ep} (= t_{j+1}^{sp}) \notin CP$ or $t_{j+1}^{st} - t_j^{et} < \sigma$, where σ is the required time for driver changing defined by a parameter.

In literature, the step of creating work-pieces is quite general as it reduces the problem size. However, in these works, work-pieces are usually constructed from a given number of tasks (mostly 2 or 3), which might result in over restricted work-pieces. For example, if the vehicle scheduling is tight, these work pieces can be quite long and it reduces the problem size in an unexpected way, producing even infeasible solutions. In our approach, instead, we consider the relations of potential consecutive tasks only, using the mentioned driver changing constraints.

Note, if we allow to change the drivers any geographical place ($CP = \mathrm{P}$), and set the required time to zero ($\sigma = 0$), each vehicle task becomes a single work-piece and we get back to the original problem.

4.2 Creating pre-shifts

Concerning the work-pieces as inputs for this first scheduling step, the optimization method creates so-called pre-shifts. A pre-shift is a sequence of pairwise compatible activities. In other words, constraints making a pre-shift legal are not necessarily considered (may be some of them) in this step. The idea is to produce quite well-optimized initial shifts in reasonable time and improve them in the following steps to feasible solutions regarding all rules. One good approach is using a qualitative optimization method. We used a network based model (Kliewer et al. 2006) applied for vehicle scheduling in normal case. Another idea can be reclining upon the present

driver shifts of the company or iterating the optimization process using the results of the previous step as pre-shifts.

4.3 Cutting

The infeasibility of the pre-shifts can be derived mostly from two reasons. One problem group is arisen by length constraints, meaning that the pre-shift is too long with respect to the constraints of maximum working time, driving time or service time. The other bunch of problems comes from the break constrains, i.e. there is no enough space inside the pre-shift for breaks as defined by the rules. Therefore, both problems can be solved by cutting the pre-shifts into smaller parts. Let $s = (t_1, \ldots, t_n)$ be a pre-shift. Define a cutting point $k (k \geq 1, k < n)$ by a given method such that $s' = (t_1, \ldots, t_k)$ is a feasible shift (i.e. fulfills every rules). Then cut s, let s' be a shift-part and continue the cutting procedure with $s'' = s/s' = (t_{k+1}, \ldots, t_n)$ until s'' itself fits to the idea of the cutting method used. The pseudo code of the algorithm is summarized below.

In the following we present the applied cutting methods.

```
Let PS=∅ be the set of the pre-shifts
while S≠∅ let s∈S, where s=(t1,…,tn)
        Define the possible cutting points in s: K = {j | 1≤j and j<n}
        if K≠∅
                Choose the best cutting point k by a cutting method (k∈K)
                Cut shift s into two part: s'=(t1,…,tk) and s''=(tk+1,…,tn)
                PS=PS∪{s'}
                S=S/{s}
                if s''≠∅
                        S=S∪{s''}
                end if
        else
                PS=PS∪{s}
                S=S/{s}
        end if
end while
```

4.3.1 Longest feasible

Considering the well arranged pre-shifts, this method cuts the longest prefix which fulfills the constraints. Formally, let $s \in S$ with $s = (t_1, t_2, \ldots, t_n)$. Then let $s' = (t_1, t_2, \ldots, t_k)$ where $k \geq n$ is the largest number such that s' is feasible. By this method one might obtain a full shift with relatively low cost, if the pre-shifts are well-optimized, but many times the postfix part is too small, making it hard to handle later. Nevertheless, if these short postfix parts can be joined to other prefixes, this method

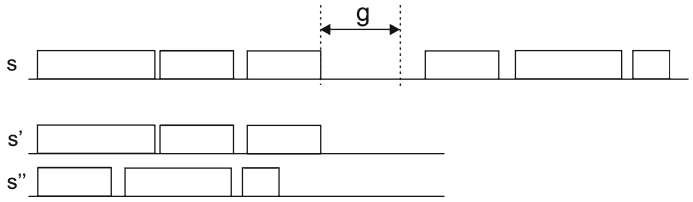

Fig. 1 Cutting pre-shifts by long gaps

works well. On the other hand, short postfix parts without appropriate long prefix pairs can result in shifts with high cost. Therefore, the success of this cutting method mainly depends on the structure of the pre-shifts.

4.3.2 Length

The idea is to cut the pre-shifts into approximately equal parts: let $s \in S$ with $s = (t_1, t_2, \ldots, t_n)$ and define the cutting point $k \leq n$ such that $\left| l - \left(t_k^{et} - t_1^{st} \right) \right|$ is minimal, where l is the defined length. This approach is the most general one, since the obtained parts can be combined in several natural ways. Note that the lengths of the parts depend on the possible cutting points in the pre-shifts, and can be even much longer or shorter than the required length. Moreover, smaller parameters l induce more difficult problems, but in this case, the result has higher flexibility for combining the shift parts.

4.3.3 Long gap

The efficiency of a shift strongly depends on the proportion of the working time and that of the idle time. In order to avoid lengthy idle time inside the shifts, this method cuts the pre-shift at an idle period longer than a given length (see Fig. 1). Let $s \in S$ with $s = (t_1, t_2, \ldots, t_n)$, and construct $s' = (t_1, t_2, \ldots, t_k)$ in such way that s' is feasible with $t_{k+1}^{st} - t_k^{et} > g$, where g is the given gap-length. If such a gap does not exist, and the shift is infeasible, then the longest feasible cutting is used. This method creates quite compact shift-parts with highly varied lengths. The length of the defined gap-length depends on the structure of the pre-shifts and the rules.

4.3.4 Objective function

This method cuts the pre-shifts at a point where the objective values of the obtained two parts are the lowest, with the condition that the first part must be feasible. Let $s \in S$ with $s = (t_1, t_2, \ldots, t_n)$, and construct $s' = (t_1, t_2, \ldots, t_k)$ such that s' is feasible and $\mathrm{c}(s')$ is minimal with $(\mathrm{c}s') + \mathrm{c}(s'') < \mathrm{c}(s)$. This approach seems to be quite useful, but on the one hand, one might obtain too small parts, and on the other hand, the objective value of a small part can be rather misleading.

4.3.5 Traffic load

During a workday the frequency of the transportation flow might be varied with high amplitude. Typically, in each city the load of the used vehicles is quite similar (cf.

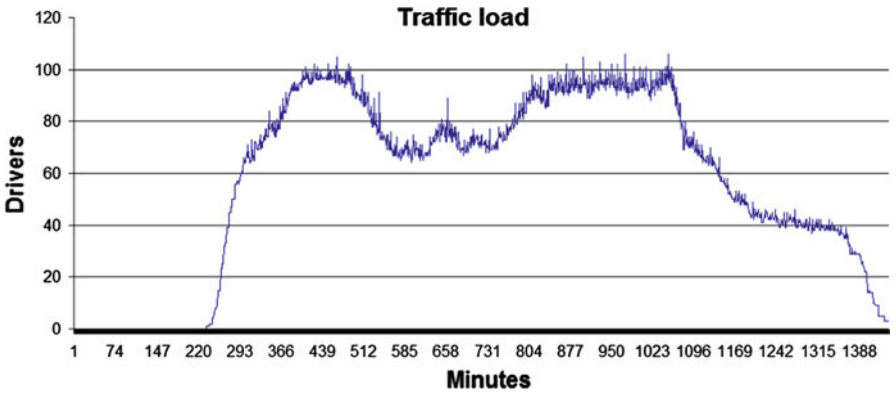

Fig. 2 Number of buses scaled by minutes in a workday of Szeged, Hungary

Békési et al. 2009) with morning and afternoon peaks(see Fig. 2). Thus, for the optimal arrangement, every driver should work in these important periods, and they should have break, rest or do other activity in the slack time. Considering this idea, the method cuts the pre-shifts such that it creates longer compact parts covering the peaks and smaller compact parts in the slack time. With these shift-parts, it is forced that all driver work in rush time and they can have breaks or rest during the other periods. Since, by the break rule, a driver needs to have break after a given working period, a long shift-part combining with the small parts in the slack periods presents the possibility for the break.

4.4 Joining

As a result of the previous step, shift-parts are considered to be joined for constructing complete shifts. Since each shift-part is feasible by itself, this joining method must keep this property. The success of this step strongly depends on the structure of the shift-parts. Moreover, this process can be time-consuming, as in every step during the method, all constraints need to be checked. One can think that the previous two steps can be skipped and the work-pieces can be joined immediately, but usually real-world problem instances are too large with a high number of work-pieces.

For measuring the quality of joining two shift-parts, a new joining-fitness function c_j is introduced. A joining is "good", if the two parts are close to each other in time (i.e. the departure time of the first trip of the second part closely follows the arrival time of the last trip of the first part), they are connected in the same release point (i.e. the departure place of the second part and the arrival place of the first part are the same) and the driver does not change vehicle (i.e. the vehicle used on the last task of the first part and the vehicle on the first task of the second part are the same). The joining-fitness function takes into consideration the above three factors by penalizing them.

Let $s_i = (t_{i1}, t_{i2}, \ldots, t_{in})$ and $s_j = (t_{j1}, t_{j2}, \ldots, t_{jm})$ be two shift-parts. Then $s' = (t_{i1}, \ldots, t_{in}, t_{j1}, \ldots, t_{jm})$ is the joint shift, and *the joining-fitness function* is calculated by

$$c^{join}(s') = \alpha * \left(t_{j1}^{st} - t_{in}^{et}\right) + \beta * x_{i,j} + \gamma * y_{i,j},$$

where

$$x_{i,j} = \begin{cases} 1 & t_{j1}^{sp} \neq t_{in}^{ep} \\ 0 & \text{otherwise} \end{cases}$$

$$y_{i,j} = \begin{cases} 1 & t_{j1}^{v} \neq t_{in}^{v} \\ 0 & \text{otherwise} \end{cases}$$

and α, β, γ are nonnegative weights.

Naturally, it is necessary to insert a new travelling activity between the two parts, if the arrival place and the departure place are different. In this step, several different methods can be applied as well, which are presented in the following.

4.4.1 Sequential

This greedy method is quite simple and fast. Order the shift-parts by their starting time, and place the first part on the beginning of the newly formed shift. Then select those parts which is compatible with the started shift (i.e. can be joined in accordance with the rules) and choose the one which can be joined with the lowest cost by the c^{join} joining-fitness function. If there is no further parts to join, close the shift under consideration, and start a new one with the next free shift-part. This process is finished when free shift-parts are consumed. The pseudo code of the algorithm is the following:

```
Let SP be the set of the shift-parts
Order SP by the departure time of the shift-parts
while SP≠∅
        Let f be the first element of SP
        SP=SP/{f}
        Select SP'⊆SP which elements compatible with f
        while SP'≠∅
                Let s'∈SP' that shift-part where c^join(f,s') minimal
                f=f+s'
                SP=SP/{s'}
                Let SP'⊆SP which elements compatible with f
        end while
        Close f
end while
```

where $f+s$ operation means the concatenation of shift f and shift-part s, with inserting a new travelling activity between them, if necessary.

4.4.2 Simultaneously

This method is also greedy, but in this case, shifts are created simultaneously. Similarly to the sequential method, order the shift-parts by their departure time, and place the first part on the beginning of the newly formed shift. Then iteratively take the next free part, select those started shifts with which this part is compatible and join to the one with the lowest c^{join} joining-fitness value. If there is no any suitable started shift, then start a new one with this shift-part. The process stops when it assigned the last shift-part. The pseudo code of the algorithm is the following:

Let SP be the set of the shift-parts
Order SP by the departure time of the shift-parts
Let F be the set of open shifts and $F=\emptyset$
while $SP\neq\emptyset$
    Let s be the first element of SP
    $SP=SP/\{s\}$
    Select $F'\subseteq F$ which elements compatible with s
    if $F\neq\emptyset$
        Let $f'\in F'$ that shift-part where $c^{join}(f',s)$ minimal
        $f'=f'+s$
    else
        $f'=s$ is a new shift
        $F=F\cup\{f'\}$
    end if
end while

where $f+s$ operation means the concatenation of shift f and shift-part s, with inserting a new travelling activity between them, if necessary.

4.4.3 Traffic load

This method is the pair of the cutting method under the same name. This joining method first defines classes labeled with lengths given by the break rule, and it classifies the shift-parts into these classes depending on their length, by considering their working

time. Then, in an appropriate order (if necessary) it makes a weighted matching on the elements of two classes, where the weights are the joining costs of the two shift-parts.

Let SP be the set of the shift-parts
Let $c=(c_1,\ldots,c_n)$ where c_i is the i^{th} working length interval by the break rule
Order c by the lengths
Classify SP by c: $C=\{C_i|s\in C_i \Leftrightarrow s\in SP$ and length of s is $c_i\}$
Let $S=C_1$ is the set of the shifts
for each C_i where $i\geq 2$
    Make a minimal weighted matching on S and C_i: M
    for each $(s,s')\in M$ where $s\in S$ and $s'\in C_i$
        $s=s+s'$
    end for
    for each $s'\in C_i$ where $s'\notin M$
        $S=\{s'\}$
    end for
end for

where $s+s'$ operation means the concatenation of shift s and shift-part s', with inserting a new travelling activity between them, if necessary.

4.5 Refining

At this point, the final (rough) shifts are generated. This method improves the objective value by changing the work-pieces between the shifts. Let s_i and s_j be two arbitrary shifts. If there exist work-pieces $t_k \in s_i$ and $t_l \in s_j$ such that moving t_k into s_j and t_l into s_i, the average cost of the two shifts is lower than the original cost, accomplish a swapping. Observe that removing or inserting a work-piece to a shift, the possible surrounding travel activities also removed or inserted. Furthermore, both shifts must remain feasible after swapping. Notice that the two swapped work-pieces are not necessarily overlapped in time. For example the last work-piece of a shift can be moved into an idle period to another shift, while the first work-piece of the second shift can be inserted into a gap to the first shift (see Fig. 3). After this swapping both shifts become more compact and have lower objective value.

This process is also required to be fast, thus a local search method was suggested instead of determining an exact optimal solution. The local search method selects

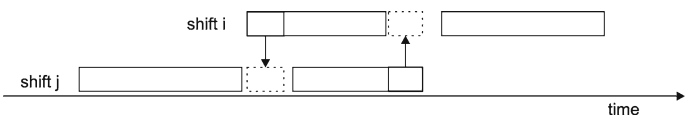

Fig. 3 Swapping two tasks between two shifts

a shift randomly and for each other shifts it searches the best swap, i.e. it chooses the shift, the cost improvement with which is the highest, and it accomplishes the operation. If there does not exist further shift with which the swapping improves the objective value, the method will not execute any operation. This process works until there is not fulfilled change for a given iteration.

In each iteration, the method chooses a random shift s_i ($s_i \in S$) and searches those shifts $s_j \in S$ ($i \neq j$), where $\exists t_k \in s_i$ and $\exists t_l \in s_j$ such that $s_i' = (s_i/\{t_k\}) \cup \{t_l\}$ and $s_j' = (s_j/\{t_l\}) \cup \{t_k\}$ are also feasible shifts with $(\mathrm{c}s_i') + \mathrm{c}(s_j') < \mathrm{c}(s_i) + \mathrm{c}(s_j)$. Then it accomplishes that swapping of t_k and t_l. If such a shift s_j does not exist, the iteration is unsuccessful. The process stops if it reaches a given number of iterations or a given number of unsuccessful iterations.

```
Let S be the set of the shifts
while not stop
        Select random si∈S
        for each sj∈S where i≠j
                Search tk∈si and tl∈sj where improve(swap(tk,tl)) maximal
                if improve(swap(tk,tl))>best_swap
                        Remember swap(tk,tl) as best_swap
                end if
        end for
        Accomplish best_swap
end while
```

Evidently, a solution with a better quality can be reached by an exact method, but according to our experiences, this approach is proved to be unacceptably slow.

4.6 Case study

The comparison of real-world driver scheduling algorithms with each other is usually not reliable: in different application areas driver scheduling methods are based on their local rules and regulations. Nevertheless, the used regulations and rules are not mentioned precisely in the literature. Several papers discuss a few basic rules only: such as the maximum (and minimum) working time and shift length, a meal break with minimum length and maximum continuous working time (Caviquel et al. 1999; Guerinik and Caneghem 1995; Wren et al. 2003; Yunes et al. 2005). Other papers concentrate merely on the selecting step of the GaS method without detailing the generating phase (Aickelin et al. 2006; Dias et al. 2002; Kwan et al. 2001; Li and Kwan 2005). Following another approach, in (Steinzen et al. 2010) a wide rule list is presented, but the rules are not specialized.

Summarizing the above facts, analyzing a solution method is most reliable inside its application area, and the best way is to compare its results to the present practice of the company.

In this section we will present a case study for showing that our solution approach with the suggested methods has the required efficiency, flexibility and two-phase property. The two-phase property is especially crucial for real-world applications: we have tested our methodology on the public bus transportation database of a middle-size city, Szeged, Hungary. We will see that appropriate completion methods can extend the results provided by our CAJ approach to real-world solutions of good quality.

The Szeged City Bus Company (Tisza Volán, Urban Transport Division) operates 40 lines served by approximately 120 buses and 160 drivers in 2,800 trips in an average working day. The frequency of the trips depends on the day of the week (working day, Saturday, Sunday or holiday) and the time of the year (summer period, school time etc.) Below we summarize the groups of main daily regulations defined by EU, the government and the transportation company, and needed to consider by CAJ and the completion phase.

a) Maximum servicing time (20 h), maximum driving time (10 h) and maximum working time (12 h) need to be considered.
b) Within 24 h (from the beginning of the shift) a daily rest of at least 12 h needs to be applied. (It can be also divided into two parts by applying certain rules)
c) The length of a break should be between 15 and 30 min and can be spent only in special places. The schedule of the breaks is prescribed by additional rules with respect to the working time (e.g. the first break needs to be applied in 6 h of working time)
d) In addition to the basic activities (driving, travelling etc.), further compulsory collateral activities need to be inserted into the shifts (work-starting and work-ending activities, administration, bus maintenance etc.)

Concerning the two-phase property, the main difficulties in applying the above rules are arising in point c) and d): the collateral activities have strong influence on the distribution of working time, thus the schedule of the breaks highly depends on them. However, collateral activities and breaks can be fixed during the completion phase only, which makes extremely hard for CAJ to produce rough shifts remaining even feasible after the completion phase (more effective solutions of rough shifts strongly reduce the possibility of completion to produce feasible solutions)

We applied our solution approach for a period of 2 months constituted by several timetable versions of the transportation company. These versions were applied for different day types (weekdays, weekend etc.). The evaluation is based on the results obtained after the completion phase, and the applied cost model is the sum of the working time in the 2 month period.

The program is coded in C++ and executed on a PC (an Intel i7 2630QM processor and 8 GB-s of memory), and for any problem instances and any methods used, the running for a 1 day schedule took less then one second. Concerning the requirement of efficiency, the above result shows the robustness of our CAJ methodology.

In the first test case the input was the vehicle scheduling of the company in the considered 2 month period. Figure 4 shows the different comparisons of the combination of the cutting and joining methods with each other and the present driver scheduling of the company.

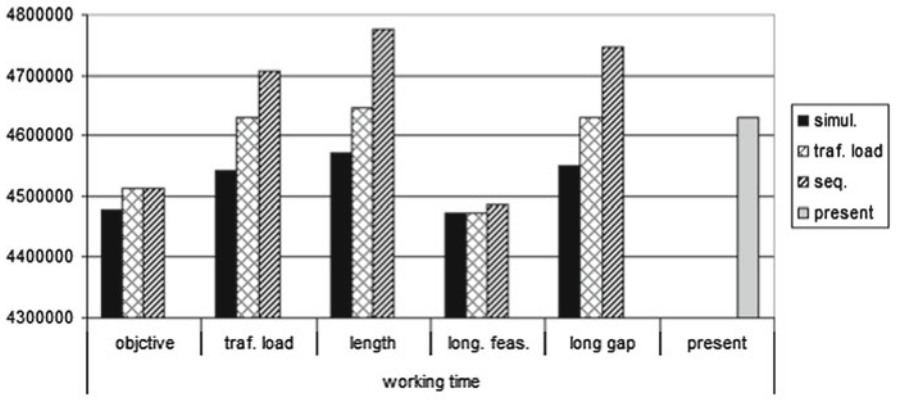

Fig. 4 Working time of 2 months by cutting and joining methods

As the comparison of the presented cutting strategies, it is clear that the "objective function" and "longest feasible" proved to be the strongest and most reliable ones. Comparing the joining strategies, it can be seen that the "simultaneously" method provided the best solution in each case. It is also clear that the "traffic load" exceeded the "sequential" strategy in almost every case. Probably the problem size and complexity makes the allocation too difficult to the latter one, though this strongly greedy method can work better with easier problem instances with keeping the fast execution. Nevertheless, the "simultaneously" strategy is the indisputable winner which is proved to be really appropriate for industrial usage.

Considering the present situation, the longest feasible strategy can guarantee a saving of 3.5 %. Note that this saving is quite significant, as approximately 2/3 of the working time constituted by the time needed by the trips, which is not possible to be reduced. Therefore, the saving on the "non-fixed" part of the schedule is more than 10 %.

Finally, in order to investigate the flexibility of our methodology, a second test case was also considered. Several times, in practical applications, a significant cost reduction in the vehicle scheduling might result in extra costs in the driver scheduling phase. Therefore, with the same conditions and evaluation principles, our algorithms were applied for an optimized vehicle scheduling provided by a time-space network approach (cf. Kliewer et al. 2006). The results are summarized on Fig. 5.

It can be seen that the longest feasible-simultaneously combination provides the best result again, and the distribution of the different solutions is quite similar to the first test case. Comparing to the previous case, each method produces a solution of lower cost, the best one guarantees a saving of 4.5 % comparing to the present situation. This result is quite significant, as the optimization on the vehicle schedule provides more than 10 % saving, thus the input of this test case is more "difficult" (vehicle schedules with high density). Therefore, the further improvement of 4.5 % proves the high flexibility of our methodology.

In summary, the results of the above test cases show the efficiency, flexibility and two-phase property of the CAJ approach.

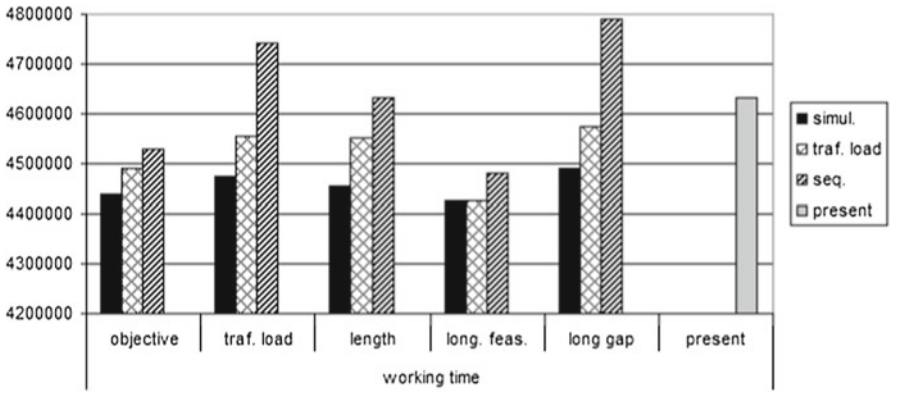

Fig. 5 Working time of 2 months by cutting and joining methods on optimized vehicle scheduling

5 Conclusion

We have presented a flexible efficient solution approach for driver scheduling. Our new methodology is based on a cut and join (CAJ) approach, which makes the method quite flexible. It turned out by case studies that the new methodology is extremely robust and flexible.

Our approach can be also used for generating fast initial solutions for more sophisticated heuristics. The flexibility of the method makes it possible to extend it to an iterative heuristic method. Our future research will concentrate on the development of such a method.

Acknowledgments This work was partially supported by the Szeged City Bus Company (Tisza Volan, Urban Transport Division) and Gyula Juhász Faculty of Education, University of Szeged (project no. CS-004/2012).

References

Aickelin U, Burke EK, Li J (2006) Improved squeaky wheel optimization for driver scheduling. In: Proceedings of the 9th international conference on parallel problem solving from nature (PPSN IX)

Békési J, Brodnik A, Pash D, Krész M (2009) An integrated framework for bus logistic management: case studies. Logist Manag 2009:389–411 (Physica-Verlag)

Cappanera P, Gallo G (2004) A multicommodity flow approach to the crew rostering problem. Oper Res 52(4):583–596

Caprara A, Toth P, Vigo D, Fischetti M (1998) Modeling and solving the crew rostering problem. Oper Res 46:820–830

Caviquel L, Rego C, Themido I (1999) Subgraph ejection chains and tabu search for the crew scheduling problem. J Oper Res Soc 50:608–616

Cheng BMW, Lee JHM, Wu JCK (1997) A nurse rostering system using constraint programming and redundant modeling. IEEE Trans Inf Technol Biomed 1:44–54

Desrochers M, Soumis F (1989) A column generation approach to the urban transit crew scheduling problem. Transp Sci 23(1):1–13

Dias TG, Sousa JP, Cuncha JF (2002) A genetic algorithm for the bus driver scheduling problem: a case study. J Oper Res Soc 53(3):324–335

Ernst AT, Jiang H, Krishnamoorthy M, Sier D (2004) Staff scheduling and rostering: a review of applications, methods and models. Eur J Oper Res 153:3–27

Guerinik N, Caneghem MV (1995) Solving crew scheduling problems by constraint programming. In: Proceedings of the 1st conference of principles and practice of constraint programming, pp 481–498

Kliewer N, Mellouly T, Suhl L (2006) A time-space network based exact optimization model for multi-depo bus scheduling. Eur J Oper Res 175(3):1616–1627

Kwan RSK, Kwan ASK, Wren A (2001) Evolutionary driver scheduling with relief chains. Evol Comput 9:445–460

Li J, Kwan RSK (2005) A self-adjusting algorithm for driver scheduling. J Heuristics 11:351–367

Nurmi K, Kyngas J, Post G (2011) Driver rostering for bus transit companies. Eng Lett 19(2):125–132

Segal M (1974) The operator-scheduling problem: a network-flow approach. Oper Res 24:808–823

Steinzen I, Suhl L, Kliewer N (2009) Branching strategies to improve regularity of crew schedules in ex-urban public transit. OR Spectr 31:727–743

Steinzen I, Gintner V, Suhl L, Kliewer N (2010) A time-space network approach for the integrated vehicle- and crew-scheduling problem with multiple depots. Transp Sci 44(3):367–382

Wren A, Fores S, Kwan A, Kwan R, Parker M, Proll L (2003) A flexible system for scheduling drivers. J Sched 6(5):437–455

Yunes T, Moura A, de Souza C (2005) Hybrid column generation approaches for urban transit crew management problems. Transp Sci 39:273–288