Hindawi Publishing Corporation
Discrete Dynamics in Nature and Society
Volume 2015, Article ID 638104, 10 pages
http://dx.doi.org/10.1155/2015/638104

Research Article

A Column Generation Based Hyper-Heuristic to the Bus Driver Scheduling Problem

Hong Li,[1] Ying Wang,[2,3] Shi Li,[4] and Sujian Li[1]

[1]*School of Mechanical Engineering, University of Science and Technology, Beijing 100083, China*
[2]*State Key Laboratory of Rail Traffic Control and Safety, Beijing Jiaotong University, Beijing 100044, China*
[3]*School of Traffic and Transportation, Beijing Jiaotong University, Beijing 100044, China*
[4]*Laboratoire Systémes et Transports, Université de Technologie de Belfort-Montbéliard, 90000 Belfort, France*

Correspondence should be addressed to Ying Wang; wangy@bjtu.edu.cn

Received 2 January 2015; Accepted 9 February 2015

Academic Editor: Chin-Chia Wu

Public transit providers are facing continuous pressure to improve service quality and reduce operating costs. Bus driver scheduling is among the most studied problems in this area. Based on this, flexible and powerful optimization algorithms have thus been developed and used for many years to help them with this challenge. Particularly, real-life large and complex problem instances often need new approaches to overcome the computational difficulties in solving them. Thus, we propose a column generation based hyper-heuristic for finding near-optimal solutions. Our approach takes advantages of the benefits offered by heuristic method since the column selection mode is driven by a hyper-heuristic using various strategies for the column generation subproblem. The performance of the proposed algorithm is compared with the approaches in the literature. Computational results on real-life instances are presented and discussed.

1. Introduction

The bus driver scheduling problem (BDSP) is an extremely complex and important part of the operational planning process of transport companies. Efficient schedules can make significant monetary savings for transportation operators since driver wages are a very large cost element. Furthermore, good crew planning ensures a high job satisfaction.

The process of bus driver scheduling involves partitioning the vehicle work into a set of legal driver shifts that should reflect the operator's definition of efficiency [1]. While the constraints according to labor rules and requirement differ from country to country, even company to company, the evaluation criteria and objectives may differ as well.

A great deal of research efforts on this problem has been made in this area since the 1960s. Many scientific approaches and systems to bus diver scheduling have been reported in a series of workshops [2–6]. Useful review and relevant review to these approaches have been given by [7]. Among these planning and scheduling processes, driver scheduling remains to be a challenging problem, and new approaches have been constantly sought to solve this problem [8].

Column generation is introduced by [9] and has been widely used in the past several years for problems in transportation, scheduling, and combinatorial optimization. The approach requires one to decompose the original problem into two problems, which are termed restricted master problem and subproblem, respectively. In the literature, column generation has been effectively applied to solve crew scheduling and other related problems (see [9–11]), particularly for real-world problems of bus driving scheduling (see [12]). However, it is well-known that one of the drawbacks of column generation is the so-called "tailing-off" effect: a lot of iterations that do not significantly modify the optimal value of the restricted master problem (RMP). Although successfully used, this method suffers from slow convergence that somewhat limits its efficiency and usability [13]. Meanwhile, hyper-heuristic is an emerging methodology in search and optimization [14]. Instead of using exact methods in most column generation, we propose a column generation based

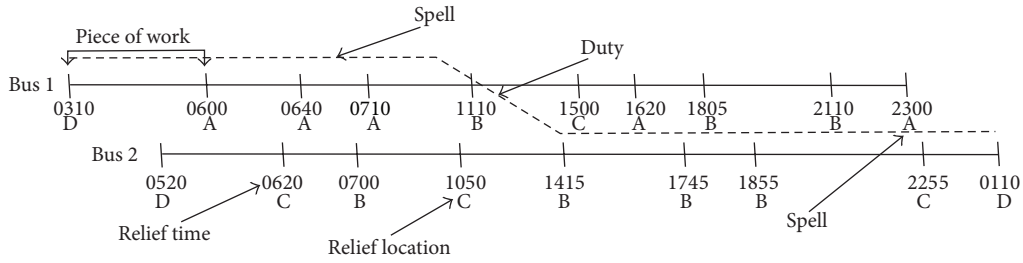

Figure 1: Relationship in vehicle block.

hyper-heuristic method for achieving good quality solutions in short running time.

This paper is organized in the following way. In Section 2, we give a problem definition of bus driver scheduling problem. A discussion of the proposed approach is given in Section 3. In Section 4, comparative results using real-life instance, some of which are large instances, are given. Finally, we conclude in Section 5.

2. The Bus Driver Scheduling Problem

2.1. Terminology. Before scheduling the drivers tasks, the vehicle routes have to be constructed. A *trip* is a movement of a vehicle in a given path. It is the basic unit of service in the sense that each trip must be operated by a single vehicle. A *vehicle block* is a sequence of trips to be done by one vehicle from the time that it leaves the depot until it returns to the depot. From the viewpoint of driver scheduling, drivers can only be relieved at some designated places called *relief points*, which are represented by letter codes (A, B, and C) in Figure 1. The times when the vehicles are at the relief points, marked on the horizontal timescale, are known as *relief opportunities*. The work between two consecutive relief opportunities on the same vehicle is called *piece of work (or task)* for the driver. The work of a driver in a day is known as a *duty (or shift)*. Note that not all relief opportunities will be used to relieve drivers and therefore a driver may be covering a number of consecutive pieces of work, called a *spell*.

An example of a duty built between two vehicle blocks is given in Figure 1. The duty is composed of two spells from the two vehicle blocks.

According to which time period the duties cover, the legal duties can be classified to the following five types.

Early Duty starts early in the morning taking the buses out of garage before the morning peak. The working time of a day that this duty covers is between 05:00 and 12:00.

Late Duty starts in the afternoon and ends in the night. The working time of a day that this duty covers is between 12:00 and 20:00.

Night Duty works in the late evening buses returning the buses to the garage. The working time of a day that this duty covers is between 22:00 and 05:00.

Day Duty starts in the morning and ends in the afternoon. The working time of a day that this duty covers is between 08:00 and 16:00.

Middle Duty works during the period of the morning and the evening peaks. The working time of a day that this duty covers has two parts. One is between 05:00 and 08:00 and the other is between 16:00 and 22:00.

The user could impose certain constraints on any of these duties to ensure that the final schedule does not contain too many or too few duties of particular type. The constrains can be imposed either initially or when reoptimizing an existing solution.

2.2. Mathematical Formulation. The set covering model is the fundamental model of the bus driver scheduling. In the public transport such models have been popular for solving crew scheduling problems for many years [9, 15]. In the formulation adopted here, we will consider our approach based on the set covering formulation of the problem. Although the schedule must ultimately have only one duty assigned to each piece of work, it is better to allow the formulation to include the possibility of overcover and to use set covering rather than set partitioning. On one hand, integer solutions may have some constraints covered by two or more nonzero variables, which in scheduling terms would mean two or more duties assigned to the same piece of work. The reason for this is that there can be a danger that there may be no feasible solution that "fit-together" without causing any over-cover. On the other hand, the produced solutions do not usually have overcovered pieces of work; the set covering model most commonly results in two duties overlapping slightly. If overcover occurs, it can be easily eliminated by editing the schedule afterwards.

Let $N = \{D_1, D_2, D_3, \ldots, D_n\}$ be the set of legal duties and $M = \{1, 2, \ldots, m\}$ be the set of pieces of work to be covered. We can define the problem as the construction of a matrix, where drivers appear in columns and pieces of works in lines. For each element $a_{ij} \in 0, 1$, $i \in M$, $j \in N$, $a_{ij} = 1$ indicates the duty j covers piece of work i, and it is 0 otherwise. It can be formulated as follows:

$$\min W_1 \sum_{j=1}^{n} c_j x_j + W_2 \sum_{j=1}^{n} x_j \tag{1}$$

subject to

$$\sum_{j=1}^{n} a_{ij} x_j \geq 1, \quad i \in M, \tag{2}$$

$$x_j \in 0, 1, \quad j \in N, \tag{3}$$

where n is the number of duties; m is number of work pieces; c_j is the cost of duty j; x_j is equal to 1 if duty j is used in the solution and 0 otherwise. W_1 and W_2 are weight constants. Generally, the main objective of the transport company is to obtain a solution with a minimum number of duties. For this objective, we consider here $W_2 \gg W_1$.

The constraints (2) indicate that each piece of work must be covered by at least one duty. We consider the costs c_j in our work based on [15]:

$$c_j = \mathrm{OV}_j + \mathrm{IT}_j, \tag{4}$$

where

$$\mathrm{OV}_j = \max\left(0, \left[\mathrm{FT}\left(t_{\mathrm{lst}}\right) - \mathrm{ST}\left(t_{\mathrm{fst}}\right)\right] - \mathrm{NWT}\right), \tag{5}$$

$$\begin{aligned} \mathrm{IT}_j = {} & \max\left(0, \mathrm{NWT} - \left[\mathrm{FT}\left(t_{\mathrm{lst}}\right) - \mathrm{ST}\left(t_{\mathrm{fst}}\right)\right]\right) \\ & + \cdots \sum_{i=1}^{m-1} \max\left(0, \mathrm{ST}\left(t_{i+1}\right) - \mathrm{FT}\left(t_i\right)\right), \end{aligned} \tag{6}$$

where c_j is the cost of the driver j; OV_j is the extra-time over the normal time of work established by law for driver j (8 hours); IT_j is the idle time of driver j (time the driver is not working during his shift); NWT is the normal working time (8 hours); $\mathrm{FT}(t_i)$ is the finish time of the task i; $\mathrm{ST}(t_i)$ is the start time of the task i; t_{fst} is the first task of the j driver's shift; t_{lst} is the last task of the j driver's shift.

3. Solution Methodology

The column generation algorithm for the BDSP is initialed by giving a small set of duties (columns). These duties constitute the initial columns in a restricted master problem (RMP). Feedback from the optimal dual values of the RMP is then used to solve a pricing subproblem, which aims to identify duties with a negative reduced cost that improves the objective function value of the RMP. The pricing subproblem for the BDSP is typically modeled as a constrained shortest path problem solved over a directed acyclic graph and then solved using dynamic programming techniques [9]. Nevertheless, because of the drawback that the subproblems for generating the columns would be computationally expensive in real problem instances [16], column generation did not make much progress toward solving large instances. Thus, the main work relies on the subproblem to speed up the whole process of column generation. Here, we propose the column generation based hyper-heuristic methodology to solve the BDSP. This approach provides two significant advantages. First, due to the fact that column generation is known for its poor convergence, it is not necessary to select only one column with lowest reduced cost; in fact, any column with a negative reduced cost will do. Using this observation can improve the overall efficiency. In our research the idea of "rapid adding columns" is to add a set of columns quickly by iteratively solving subproblem using a set of low heuristics. Thus, more than one duty with negative reduced costs may even be possible to be brought into the restricted subset per iteration. Second, the diverse columns with negative reduced cost that are selected (or generated) can be performed efficiently, see [13], to avoid the rapid and useless increase in RMP size. In fact, our approach may select a good "diversity" of duties by using these low heuristics to improve the solution without a large increase in computation time. The pricing step plays an important role in terms of the efficiency of the column generation method.

An outline of the algorithm is given as follows.

Step 0 (preprocessing). Generate all valid duties. Then, these duties form a column pool P.

Step 1 (construct an initial solution). Take a small subset of the duties $p \subset P$ as initial set of columns.

Step 2 (solving the RMP). Solve the LP over the current duty subset and compute the shadow prices of the set of columns p.

Step 3 (duty management). Control the column pool P and remove the columns in the RMP if necessary.

Step 4 (selection of new duties). Select duties with negative reduced cost which will improve the solution, and add them to the RMP.

Step 5 (stopping criterion). If the stopping criterion is met, then go to Step 6; else go to Step 2.

Step 6 (finding integer solutions). Solve ILP and obtain an integer solution.

The overall execution is iterated between Steps 2 and 6. In the first iteration it uses the columns created in Step 0.

In the remainder of this section, we provide details of each of the above steps.

3.1. Step 0: Preprocessing. In our approach we generate all feasible duties explicitly in a preprocessing step. That is, all the possible combinations of pieces of work are valid according to a set of labor contracts and regulations. One of the advantages of generating duties beforehand is that different duty sets can be identified so that experienced schedulers can "share" their own knowledge rather than relying on a "black box" to produce the schedules. Error correction or experimental changes can thus be carried out relatively quickly before searching for resulting schedules [17]. Moreover, preprocessing is an important procedure to express "preferences" and "undesirableness," which are defined by a user-defined strategy. As discussed in [16], soft parameters are used to limit the generation of duties, for example, limiting the maximum number of spells in a duty to four. Meanwhile,

(1) Given a bus schedule, a maximum working time of duty T, and a required number of pieces N_r; Set $N_p = 1$, $j = 1$
(2) **for** all bus vehicles V **do**
(3) Select the relief opportunities to generate a set of piece of work $M = \{M_1, M_2, \ldots, M_m\}$
(4) Find the piece $M_i \in M$ such that the length is shortest (denoted as $t_{\min}$) in M
(5) Compute a maximum number of piece $N_{\max} = T/t_{\min}$
(6) **end for**
(7) **while** $N_p \leq \text{Min}(N_r, N_{\max})$ **do**
(8) **for** piece of work $M_{[j][i]} \in M$ **do**
(9) **for** piece of work $M_{[j][i+1]} \in M$ **do**
(10) $\cdots$
(11) **for** piece of work $M_{[j][N_p]} \in M$ **do**
(12) **if** Duty D_i which covers $(M_{[j][i]}, M_{[j][i+1]}, \ldots, M_{[j][N_p]})$ satisfies the constrains (9)–(13) **then**
(13) Add generated duty i to set of duties D
(14) N_p++; j++
(15) **end if**
(16) **end for**
(17) **end for**
(18) **end for**
(19) **end while**1111eq:eps

Algorithm 1: Generation of all valid potential duties.

a set of preferred duties can be identified relatively quickly; for example, duties with idle time less than average are preferred. Here, this kind of preprocessing could help us to find a subset of "good" duties when we need to use them during the search process.

Formally, we make the following definitions.

Definition 1. Given a duty D_j, let R_i^j denote a relief opportunity which is start point in a piece of work i covered by the duty D_j. For a given relief opportunity R_i^j, we define a relief time and relief point in R_i^j as $\text{ST}(t_i)$ and $\text{AS}(t_i)$, respectively. Then, $R_i^j = (\text{ST}(t_i), \text{AS}(t_i))$.

Definition 2. Given a duty D_j, let L_i^j denote a relief opportunity which is end point in a piece of work i covered by the duty D_j. For a given relief opportunity R_i^j, we define a relief time and relief point in L_i^j as $\text{FT}(t_i)$ and $\text{DS}(t_i)$, respectively. Then, $L_i^j = (\text{FT}(t_i), \text{DS}(t_i))$.

Based on the above definitions, a piece of work i covered by the duty D_j is denoted by

$$M_{ji} = \left(R_i^j, L_i^j\right). \tag{7}$$

Hence, n pieces of work composed of a duty D_j can be defined as

$$D_j = \left(M_{j1}, M_{j2}, \ldots, M_{jn}\right). \tag{8}$$

According to Definitions 1 and 2, each duty is forced to satisfy all the following basic constraints:

$$\left[\text{FT}\left(t_{\text{lst}}\right) - \text{ST}\left(t_{\text{fst}}\right)\right] \leq \text{MWT}, \tag{9}$$

$$\sum_{t_i \in D_j} \left(\text{FT}\left(t_i\right) - \text{ST}\left(t_i\right)\right) \leq \text{MVT}, \tag{10}$$

$$\text{FT}\left(t_i\right) \leq \text{ST}\left(t_{i+1}\right), \tag{11}$$

$$\text{AS}\left(t_i\right) = \text{DS}\left(t_{i+1}\right), \tag{12}$$

$$\begin{aligned} &\text{AS}\left(t_i\right) \neq \text{DS}\left(t_{i+1}\right), \\ &\text{ST}\left(t_{i+1}\right) - \text{FT}\left(t_i\right) \geq T_{i,i+1}, \end{aligned} \tag{13}$$

where MWT is the maximum working time (8 hours + 2 hours); MVT is the maximum valid working time; $\text{AS}(t_i)$ is an arrival station of task i; $\text{DS}(t_{i+1})$ is a depart station of task $(i + 1)$; $T_{i,i+1}$ is the necessary time to reach $\text{DS}(t_{i+1})$ from $\text{AS}(t_i)$.

Here we propose a method to generate the valid potential duties as follows in Algorithm 1. The algorithm is initialized by setting a number of pieces N_p and i equal to 0. The relief opportunities are selected to generate a set of all the possible pieces of work M according to a given bus schedule. Notice that a maximum number of pieces, which is limited to compose a duty, are different in some practical cases, for example, no more than four spells in [18]. Another example is from [19] and in their case the maximum is equal to 2. Hence, we will refer to a required number of pieces given by the bus company as N_r. Let us denote $t_{\min}$ corresponding to the shortest length of piece in M (line (4)). Then, we can compute a maximum number of pieces $N_{\max}$ (maximum working time of duty divided by the shortest time of piece) on line (5). N_p is thus determined by N_r and $N_{\max}$. Finally, the process adds one piece of work after one piece of work into a duty until the maximum number of pieces is met. Nested loops on lines (7)–(19) are iterated gradually through all pieces of work of each vehicle. Line (12) checks if the duties satisfy the constrains (9)–(13) before adding the duties to the pool P.

3.2. Step 1: Construct an Initial Solution. For computing initial shadow prices, we propose a greedy constructive heuristic.

(1) Initialize $Q \leftarrow \emptyset$
(2) **while** $Q \neq M$ **do**
(3) Choose $D_j \in S$ with minimize cost c_j'
(4) $Q_{k+1} = Q_k \cup D_j$
(5) **end while**
(6) Find an initial solution to BDSP

Algorithm 2: Greedy constructive heuristic.

An initial solution is made by a sequential mechanism, whose main objective is to obtain initial values of dual variables to select more duties and improve the linear solution in Step 2. The procedure is presented in Algorithm 2. Given a set of legal duties $S = \{D_1, D_2, D_3, \ldots, D_n\}$ and a set of pieces of work to be covered $M = \{1, 2, \ldots, m\}$, let Q be a set of the pieces of work covered so far. As it is a constructive heuristic, a new duty not yet assigned is added in a greedy way to the current schedule at each iteration. With this objective, we calculate a new cost of duty c' as a greedy function for our choice at each iteration. The new cost of duty c_j' can be computed as the following:

$$c_j' = \frac{c_j}{\sum_{i=1}^{m} \beta_{ij}}, \quad j = 1, 2, \ldots, n, \tag{14}$$

where c_j is the cost of duty D_j in the rth iteration; $\beta_{ij} = 1$ if not yet covered piece of work i is covered by D_j after the rth iteration, and 0 otherwise. Particularly, if $\sum_{i=1}^{m} \beta_{ij} = 0$, $c_j' = c_j$.

Finally, the process is repeated until all pieces of work are covered.

3.3. Step 2: The Restricted Master Problem. Following a typical column generation, the LP relaxation of the IP is necessary to be applied; that is, we relax the integer constraints (4). Thus, constraints (4) can be rewritten as

$$x_j \in R, \quad j \in N. \tag{15}$$

The purpose of solving RMP is to find the dual prices, or shadow prices, corresponding to the pieces of work in the problem. These values are used to calculate the reduced cost of new duties described in Step 4.

3.4. Step 3: Duty Management. Duty management (or column management) is an integral part of any successful column generation algorithm as discussed in [18–20].

In this research, there are two reasons why this process is applied.

Firstly, at each column generation iteration we insert all columns with negative reduced costs that have been found by the pricing algorithms into the RMP (see Step 4). Thus, it is clear that the procedure to RMP is not efficient if there are more columns than a certain maximum amount. In our test, all the duties with reduced cost greater than a given threshold will be removed whenever the number of duties exceeds 50,000. Similar strategies are used in [21]. This process is particularly important when the RMP keeps growing enormously during the column generation process.

Secondly, the selection of duties (see Step 4) can sometimes be sped up without loss of quality if the number of duties generated can be reduced intelligently [18]. Since the number of duties for some real-life problems may be quite large, we consider to take measures to prevent "bad" quality of duty into our procedure. The following procedure is iterated as long as the number of duties stays above 100,000. Each potential duty will be ranked by combining two attributes: an index which reflects its apparent efficiency (driving time divided by working time) and a ratio which reflects the identical duties; for example, a ratio of duty which is wholly contained by another duty is 0. Then, the lowest ranked duties are discarded until some definitive conditions are met. The removal of these duties reduces the overall size of a subproblem, which reduces run-time and memory usage requirement.

3.5. Step 4: Selection of New Duties. The major component of column generation is the procedure for selecting (generating) candidate duties to bring into the restricted subset in each iteration. The goal is to improve the LP relaxation of the restricted master problem. Thus, the objective of this subproblem is to select the negative reduced cost of duties not already in the restricted subset. We now provide the details of this procedure to identify duties with negative reduced costs at each iteration. The reduced cost is obtained from the cost and dual values j of each corresponding set covering constraint as given in the previous section. As only negative reduced-cost routes can enter the restricted master problem, the reduced cost should be minimized or constrained to be negative. Using the dual variables from the LP solution, the reduced cost of each duty is calculated by

$$\widetilde{c} = c_j - \sum_{i=1}^{m} \pi_i a_{ij}, \quad j = 1, 2, \ldots, n, \tag{16}$$

where $\widetilde{c}$ is the reduced cost of duty (column) j; c_j is cost of duty j; π_i is the dual price of task i.

In this research, we develop a heuristic method based on the idea of hyper-heuristic that, while efficient, is not guaranteed to identify all the columns. To speed up the search for columns with negative reduced cost, we perform heuristic pricing. Besides saving computing time, this also allows generating multiple columns in each run. For one heuristic method is greedy, by applying different heuristic methods, we can keep the diversity of columns and avoid getting into a local optimal direction. Based on the framework of hyper-heuristic [14, 22], the competition rules are applied to guide the selection of the low-level heuristic during the search process. When a low-level heuristic is applied, the performance of this low-level heuristic will be selected by a function Δ. To prevent holding a large computation time by a poor performance of low heuristic, the simple ranking [23] is used to exclude the worst heuristic according to its scores in the last application at each iteration. There are four different cases (shown in Table 1) to reflect the important performance of a low-level heuristic.

(1) Determine properties of each duty: number of spells, duty type and cost.
(2) Construct different neighborhood structures.
(3) Begin with one neighborhood structure S_i' which is randomly selected to search.
(4) **while** countmove < Maxumummove **do**
(5) Candidate duty D_i is chosen randomly in S_i'
(6) $D_{\text{incumbent}} \leftarrow D_i$
(7) $D_{\text{current}} \leftarrow$ Apply 2-exchange
(8) **if** current duty is better than incumbent **then**
(9) $D_{\text{incumbent}} \leftarrow D_{\text{current}}$
(10) **end if**
(11) countmove++
(12) **end while**
(13) Add $D_{\text{incumbent}}$ to RMP if it has a negative reduced cost

Algorithm 3: Local search.

Table 1: Reward in the particular cases.

Case	Description	Score
Case 1	A negative reduced cost of duty found is the minimum among others.	3
Case 2	The execution time is less than other heuristics.	2
Case 3	The CPU seconds elapsed is much more than other heuristics since it was last called.	1
Case 4	No duty with negative reduced cost found.	−1

At each iteration, Δ of each low-level heuristic is calculated by

$$\Delta = \sum_{k=1}^{4} r_k, \tag{17}$$

where r_k is the score of case k.

Note that if each low-heuristic has the same total score at given iteration, no low-heuristic will be excluded. Five heuristics select new duties in the same time at the beginning.

3.5.1. Swap Heuristic. The authors in [24] consider that if a column in the optimal solution obtained by Step 2 is modified appropriately, it is likely to get a new column with negative reduced cost. For this reason, we try to select a number of columns in this heuristic.

Swap heuristic is formally written as follows. Given a duty D_j from optimal solution $S = \{D_1, D_2, \ldots, D_n\}$ of the restricted master problem, we decompose the duty D_j that is randomly selected in partial consecutive duty into h, k pieces of work (POW). Moreover, assuming that $h \leq k$ and $h + k = n$, then D_j is decomposed as $D_{j1} = \cup_{i=1}^{h}\text{POW}$, $D_{j2} = \cup_{i=h+1}^{n}\text{POW}$. Therefore, all the duties which contain either $D_{j1} = \cup_{i=1}^{h}\text{POW}$ or $D_{j2} = \cup_{i=h+1}^{n}\text{POW}$ can be found in P (see in Step 0). Thus, we can select the duty with negative reduced cost to RMP. This process is executed until either at least one new duty with negative reduced cost can be found or a predefined number of attempts without success is reached.

3.5.2. Local Search Version 1. As we know, the element of search space consists of various duties. Thus, we could consider several neighborhood relations according to the properties of a duty, that is, duty type, number of spells, and cost. Then, one of neighbors is randomly selected in order to obtain an enough "good" duty after applying a local search. In the following, we will describe a neighborhood relation that we consider in our case.

It was mentioned earlier that five types of a duty are specified: Early Duty, Late Duty, Night Duty, Day Duty, and Middle Duty. Let $S_i' = (D_1', D_2', \ldots, D_n')$ be a subset of duties that each duty $D_i' \in S_i'$ is performed in the same duty type. As illustrated in the pseudo code of Algorithm 3, the neighborhoods S_i' allow the search to move (Step (3) of Algorithm 3). Then, the local search starts from a duty D_i by using 2-exchange, which follows a random sequence of duties in S_i' (Steps (5)–(7) of Algorithm 3). If D_{current} with less reduced cost is found, $D_{\text{incumbent}}$ will be replaced (Step (8) of Algorithm 3). All the above steps will be repeated until the predefined number of iterations is reached.

3.5.3. Local Search Version 2. This heuristic is similar to *local search version* 1, except the neighborhood selection mode that a greedy strategy is used. We define a set of neighborhoods used according to the number of spells. Thus, these different neighborhoods can be ranked in a descendant order. The duty with the maximum number of spells will be checked at each iteration. However, another neighbor will be selected according to the rank if a duty with negative reduced cost cannot be found in this neighbor after 10 attempts without success. And so on like this way to search, the entire search stops when a suitable stopping condition is met (here we use a maximum allowed number of iterations based on the CPU time).

3.5.4. A Greedy-Based Heuristic Version 1. Instead of searching the whole space, the first heuristic attempts to drive the search towards definitive zone of space. We maintain a reasonable size of the duties which are deemed to search efficiently (i.e., a limited number of duties $N_{\text{limit}} \subset N$). Then we try to find a "desired" duty in this subset.

More exactly, let P_{cost} be a pool which only stores the duties in which the costs are less than an adaptive threshold. This threshold is dynamically controlled by a parameter η. Then, the duties are ordered in a descendant order according to their costs. Duties, therefore, will be checked one by one according to this ranking order until one of some stopping criteria is met, which is either a time limit or finding one duty with negative reduced cost. In fact, the heuristic cannot find a "desired" duty after a certain time when entire search is stopped; it perhaps means that the threshold should be increased. Here we define that the threshold will be updated by $\min(c_j)$.

3.5.5. A Greedy-Based Heuristic Version 2. This heuristic is similar to a greedy-based heuristic version 1, except that a duty which is selected randomly in the pool P will be checked at each iteration.

3.6. Step 5: Stopping Criterion. In the context of this work, two stopping criteria are defined in our approach: (1) no duty with negative reduced cost is found and (2) the time limit is reached.

3.7. Step 6: Finding Integer Solutions. As mentioned earlier, column generation is used to solve the relaxed master problem. Hence, the resulting LP solution produced a solution with fractional numbers of duties and an integer solution must be found. In this case, a branch and bound procedure is used in order to deliver an integer solution. Several branching strategies compatible with column generation techniques are discussed [21, 25]. Constraint branching, which was originally proposed in [26], is widely used for branching strategies. In theory, constraint branching is used to set partitioning models and could lead to suboptimality if applied to inequality constraints, but since the optimal integer solution to the current problem will not necessarily equate to the optimal schedule and users can seldom specify what they mean by optimality, the emphasis is on finding a good integer solution quickly. Formally, let $J(s,t)$ be a set of variables from the optimal fractional solution in the RMP, where each variable covers tasks s and t simultaneously and $\sum_{j\in J(s,t)} x_j$ as the sum of solution values of the variables in the set $J(s,t)$. In [26], any optimal fractional solution contains at least one constraint pair s, t, for which $\sum_{j\in J(s,t)} x_j$ lies strictly between zero and one (see a proof in [27]):

$$0 < \sum_{j\in J(s,t)} x_j < 1. \tag{18}$$

The equivalent of the zero branch and the one branch are, respectively,

$$\sum_{j\in J(s,t)} x_j \le 0, \tag{19}$$

$$1 \le \sum_{j\in J(s,t)} x_j. \tag{20}$$

Applying the above branching strategy on our problem, branching is done by dividing the solution space into two sets. The branching can be imposed on the 0-branch:

$$\sum_{j\in J(s,t)} x_j = 0. \tag{21}$$

The 0-branch forces all duties covering both tasks are banned. Then on the 1-branch,

$$1 \le \sum_{j\in J(s,t)} x_j. \tag{22}$$

The 1-branch implies that any duty which covers r_1 (or r_2) but not r_2 (or r_1) must be excluded.

Instead of solving the linear program to optimality, we can choose to prematurely end the process if the relative gap between upper bound and the lower bound is smaller than ε. Upper bound on the objective value is given by the integer solutions found.

A lower bound can be obtained by Step 3.

LB is the lower bound on the optimal solution value computed by solving the LP relaxation of (1) and rounding up the corresponding value. Column generation iterations can be terminated when this gap is less than a user-defined tolerance, giving a near-optimal solution of defined quality.

Definition 3. T_b is the total running time of bus, which can be computed from a bus schedule.

Assuming that the users predefine MWT as the maximum working time of a driver per day, N_E given to estimate the number of drivers expected is calculated by

$$N_E = \left\lceil \frac{T_b}{\text{MWT}} \right\rceil. \tag{23}$$

Proposition 4. *N_E is a tight low bound.*

Proof. Given a bus schedule, T_b is constant. Let $D_S = (d_1, d_2, \ldots, d_n)$ be the set of duties in an optimum solution S. Note N_S is a number of drivers in S.

Case a. the effective working hours of duty $d \in D_S$ are the maximum working time MWT. Therefore,

$$N_E = N_S = \left\lceil \frac{T_b}{\text{MWT}} \right\rceil. \tag{24}$$

Case b. It has occurred when there is at least one duty $d_i \in D_S$ such that the effective working hours are not equal to MWT:

$$N_E < N_S = \left\lceil \frac{T_b}{\text{MWT}} \right\rceil + n, \tag{25}$$

where $1 \le n$ and integer.

Hence N_E is a tight low bound. □

The process continues until the gap is less than a user-defined tolerance, giving a near-optimal solution of defined quality. However, this could be time consuming for large problems. Therefore, the branching process will terminate the search once one integral solution has been found if a given execution time is exceeded.

Table 2: Comparative results against the best known schedules.

Data	Number of pieces of work	Number of legal duties	Best known solutions	
			Number of duties in schedule	Time (seconds)
A1	25	1,006	12	0.10
A2	50	10,552	20	4.63
A3	100	42,473	40	1.88
A4	250	990,065	81	70.90
A5	500	8,388,608	145	6567.80
B1	200	17,131	25	—
B2	265	19,835	73	—

4. Computational Results

In this section, we present the results obtained for the proposed approach on several real-world instances. These instances vary in size and complexity. The instances, denoted by A1–A5in Table 2, are from Mauri & Lorena [15]. For the last two instances, the test data comes from two lines of bus in Beijing. Table 2 shows the size (pieces of work) and the best known results of real-world problems.

In order to evaluate the quality of the solution obtained by our approach presented here, the measure of the quality of a heuristic schedule is shown as follows:

$$\text{Devation} = \left(\frac{\text{result} - \text{bestknown}}{\text{bestknown}}\right) \times 100\%. \quad (26)$$

All computational experiments were performed in a personal computer with Intel Core 2 Duo CPU T5870 that is processor of 2.00 GHz, 1.99 GHz with 2.00 G of RAM memory on Microsoft Windows 2002. The whole implementation was developed in the C # language and has been complied using the NET framework version 2.2.30729. The ILP solver used in experiments is LINGO 11.0. This software has been used to solve RMP after each pricing iteration. The threshold has been set to 5 min $\tilde{c}$ in Step 3. The computation time to explore search space in the subproblem is limited to 2 hours (3,600 seconds).

The results of Table 3 show that in most of cases of instances A1–A5 our method could find solutions which are the same as the best known solutions. We can also find that our method does not perform well in some small size instances like A1–A3 compared to PTA/LP and TA/LP, even to SA. However, as the size of instance increases, our method performs better. This is because that, for small instance, our method is too complex and its computing time is long. But for large instance, the classic methods meet the problem of combinational explosion which is the typical feature of NP problem, while our method begins to show its advantage of saving computing time.

In Table 4 we present the "manual" solutions of last two instances currently used in the company and the solutions obtained with our approach. Our solutions represent a reduction of −8% and −8.22%, respectively, on the manual solutions. This improved schedule was very acceptable to the user.

Table 3: Comparative results against the best known schedules.

Problem	Used method	Number of duties	Deviation%	Times
A1	PTA/LP[a]	12	0	0.10
	TA/LP[b]	12		0.30
	SA[c]	12		1.90
	SA_20[d]	12		35.48
	CGBH[e]	12		10.58
A2	PTA/LP	20	0	4.63
	TA/LP	20		41.64
	SA	20		42.87
	SA_20	20		949.04
	CGBH	20		39.53
A3	PTA/LP	40	2.5	1.88
	TA/LP	40		4.98
	SA	40		7.60
	SA_20	40		173.26
	CGBH	41		150.25
A4	PTA/LP	81	1.23	70.90
	TA/LP	82		229.68
	SA	85		199.86
	SA_20	83		3749.02
	CGBH	82		1309.07
A5	PTA/LP	145	6.89	6567.80
	TA/LP	151		6717.30
	SA	153		7061.52
	SA_20	153		143565.20
	CGBH	155		4038.13
Average			2.12	

[a]Population training algorithm/linear programming with more iterative.
[b]Population training algorithm/linear programming with few iterative.
[c]Simulated annealing.
[d]Simulated annealing with 20 executions.
[e]Colunm generation based hyper-heuristic approach.

5. Conclusion

We describe a methodology for finding near-optimal solutions to bus driver scheduling problem. Suitable combination of column generation and hyper-heuristic can benefit much from synergy and exhibit higher performance with respect to solution quality and time for some cases, whereas the combination of column generation and hyper-heuristic also

Table 4: Comparison with manual solutions.

Type of Duty	B1			B2		
	Manual Solution	Our Solution	Deviation%	Manual Solution	Our Solution	Deviation%
Early Duty	8	6		27	25	
Late Duty	8	2		27	19	
Day Duty	0	4		0	6	
Middle Duty	9	11		19	17	
Total Duty	25	23	−8	73	67	−8.22

needs substantial further research in order to make them fully developed. Our future work on such hybrid approach will be still continued.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgment

This work is supported by the Fundamental Research Funds for the Central Universities (2014JBM072).

References

[1] H. R. Lourenço, J. P. Paixão, and R. Portugal, "Multiobjective metaheuristics for the bus-driver scheduling problem," *Transportation Science*, vol. 35, no. 3, pp. 331–343, 2001.

[2] A. Wren, *Computer Scheduling of Public Transport*, North-Holland Publishing, Amsterdam, The Netherlands, 1981.

[3] M. Desrochers and J. M. E. Rousseau, *Computer Scheduling of Public Transport*, Elsevier, Berlin, Germany, 2009.

[4] J. R. Daduna, I. Branco, and J. M. P. Paixão, *Computer-Aided Transit Scheduling: Proceedings of the Sixth International Workshop on Computer-Aided Scheduling of Public Transport*, vol. 430, Lecture Notes in Economics and Mathematical Systems, 1995.

[5] M. D. Hickman, P. B. Mirchandani, and S. Voss, *Computer-Aided Systems in Public Transport*, Springer, Berlin, Germany, 2008.

[6] S. Voss and J. R. Daduna, *Computer-Aided Scheduling of Public Transport*, Springer, 2001.

[7] A. Wren and J. Rousseau, *Bus Driver Scheduling—An Overview*, Springer, Berlin, Germany, 1995.

[8] L. Zhao, "A heuristic method for analyzing driver scheduling problem," *IEEE Transactions on Systems, Man, and Cybernetics Part A: Systems and Humans*, vol. 36, no. 3, pp. 521–531, 2006.

[9] M. Desrochers and F. Soumis, "A column generation approach to the urban transit crew scheduling problem," *Transportation Science*, vol. 23, pp. 1–13, 1989.

[10] S. Lavoie, M. Minoux, and E. Odier, *A New Approach of Crew Pairing Problems by Column Generation and Application to Air Transports*, Université de Montréal, Centre de Recherche sur les Transports, 1985.

[11] T. Nishi, Y. Muroi, and M. Inuiguchi, "Column generation with dual inequalities for railway crew scheduling problems," *Public Transport*, vol. 3, no. 1, pp. 25–42, 2011.

[12] S. Fores, L. Proll, and A. Wren, *A Column Generation Approach to Bus Driver Scheduling*, Elsevier, 1996.

[13] N. Touati Moungla, L. Létocart, and A. Nagih, "Solutions diversification in a column generation algorithm," *Algorithmic Operations Research*, vol. 5, no. 2, pp. 86–95, 2010.

[14] E. Burke, G. Kendall, J. Newall, E. Hart, P. Ross, and S. Schulenburg, "Hyper-heuristics: an emerging direction in modern search technology," in *Handbook of Metaheuristics*, vol. 57 of *International Series in Operations Research & Management Science*, pp. 457–474, Springer, Berlin, Germany, 2003.

[15] G. R. Mauri and L. A. N. Lorena, "A new hybrid heuristic for driver scheduling," *International Journal of Hybrid Intelligent Systems*, vol. 4, pp. 39–47, 2007.

[16] R. S. K. Kwan and A. Kwan, "Effective search space control for large and/or complex driver scheduling problems," *Annals of Operations Research*, vol. 155, no. 1, pp. 417–435, 2007.

[17] S. Fores, L. Proll, and A. Wren, "Experiences with a flexible driver scheduler," in *Computer-Aided Scheduling of Public Transport*, S. Vo and J. R. Daduna, Eds., vol. 505 of *Lecture Notes in Economics and Mathematical Systems*, pp. 137–152, Springer, Berlin, Germany, 2001.

[18] A. Wren, S. Fores, A. Kwan, R. Kwan, M. Parker, and L. Proll, "A flexible system for scheduling drivers," *Journal of Scheduling*, vol. 6, no. 5, pp. 437–455, 2003.

[19] D. Huisman, R. Freling, and A. P. M. Wagelmans, "Multiple-depot integrated vehicle and crew scheduling," *Transportation Science*, vol. 39, no. 4, pp. 491–502, 2005.

[20] İ. Mutera, Ş. İ. Birbila, K. Bülbüla et al., "Solving a robust airline crew pairing problem with column generation," *Computers & Operations Research*, vol. 40, no. 3, pp. 815–830, 2013.

[21] F. Liberatore, G. Righini, and M. Salani, "A column generation algorithm for the vehicle routing problem with soft time windows," *4OR*, vol. 9, no. 1, pp. 49–82, 2011.

[22] E. K. Burker, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. R. Woodward, "A classification of hyper-heuristic approaches," in *Handbook of Metaheuristics*, M. Gendreau and J. Potvin, Eds., vol. 146, pp. 449–468, Springer, Boston, MA, USA, 2010.

[23] E. Burke, T. Curtois, M. Hyde et al., "Iterated local search vs. hyper-heuristics: towards general-purpose search algorithms," in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC '10)*, pp. 1–8, Barcelona, Spain, July 2010.

[24] Z.-L. Chen and H. Xu, "Dynamic column generation for dynamic vehicle routing with time windows," *Transportation Science*, vol. 40, no. 1, pp. 74–88, 2006.

[25] M. Mesquita and A. Paias, "Set partitioning/covering-based approaches for the integrated vehicle and crew scheduling problem," *Computers and Operations Research*, vol. 35, no. 5, pp. 1562–1575, 2008.

[26] D. M. Ryan and B. A. Foster, "An integer programming approach to scheduling," *Computer Scheduling of Public Transport*, vol. 1, pp. 269–280, 1981.

[27] C. Barnhart, N. L. Boland, L. W. Clarke, E. L. Johnson, G. L. Nemhauser, and R. G. Shenoi, "Flight string models for aircraft fleeting and routing," *Transportation Science*, vol. 32, no. 3, pp. 208–220, 1998.