Engine Routing and Scheduling at Industrial In-Plant Railroads

Marco E. Lübbecke • Uwe T. Zimmermann
Department of Mathematical Optimization, Braunschweig University of Technology, Pockelsstraße 14, D-38106 Braunschweig, Germany
m.luebbecke@tu-bs.de • u.zimmermann@tu-bs.de

In-plant railroad engine scheduling involves routing and scheduling decisions for a heterogeneous fleet of switching engines to serve a set of time-window- and capacity-constrained transportation requests. Despite an ever-increasing competition, the current planning is purely by pencil and paper. Our paper describes the mathematical and algorithmic developments for addressing in-plant railroad decision support for scheduling and routing. The problem discussed in our work is related to the multiple-vehicle pickup and delivery problem. Exploiting the structure of admissible schedules of our particular railroad situation, we introduce two formulations of the problem as mixed integer and set partitioning programs. We propose solving the linear programming relaxation of the set partition model by column generation. We focus on the pricing problem stated in the form of a constrained shortest path problem, which is $\mathcal{NP}$ complete in the strong sense. A new exact label correcting algorithm is developed that prunes the search space in a novel manner. Heuristically obtained integer solutions of a practical quality are proposed as well. All the claims are demonstrated by computational experiments on both artificial and real-life data. We discuss implementation details as well.

Introduction

Large industrial plants in the chemical, automobile, and steel industry often occupy entire quarters of cities. Heavy freight must be transported between widely spread terminals. To maintain a timely around-the-clock production process, it is often indispensable to operate a private industrial railroad. With few exceptions (Charnes and Miller 1956), there has been no extensive discussion of in-plant railroad scheduling in the operations research literature. However, ongoing privatization and market deregulations in the railroad sector require a better transportation quality at reduced charges because truck transport is the obvious competition. The need for an efficient employment of existing resources, especially locomotives, and the resulting demand for a computer-aided scheduling system is widely recognized, motivating the present research.

In-plant railroad operation is strictly customer oriented, and not based on train schedules. Terminals submit requests for empty freight cars and specific materials. These requests are addressed by a fleet of switching engines. The latter differ in their personal and technical equipment and perform all coupling, switching, weighing, and decoupling of trains; they need fueling, maintenance, and repair. Engineers have to have breaks, shift changeovers must take place, and safety regulations must be observed. This incomplete list gives an impression of the variety of tasks that are to be scheduled, usually with respect to given time windows. In the next section we describe the operational constraints in more detail. The goal is to maximize productivity of the engines.

Current planning tools merely help in gathering and displaying information. The actual disposition, however, is done manually; the decision about

0041-1655/03/3702/0183$05.00
1526-5447 electronic ISSN

how to schedule engines is completely up to the dispatcher and his or her experience and motivation. Planning ahead is practically impossible during peak workloads. Requests are scheduled on a first-come, first-served basis. Clearly, a human judgment of dependencies between interwoven decisions of such complexity is by necessity local and incomplete. We develop a mathematical model, and appropriate algorithms for a provably good solution, providing an *active* planning support for workaday operations.

Our paper is organized as follows. We formally introduce the *engine scheduling problem* in §1. A mixed integer and a binary set partitioning formulation, respectively, are developed in §2. We solve the linear programming relaxation of the latter model by column generation in §3. In §4, we focus on the pricing problem, and propose an exact label correcting algorithm as well as heuristic ideas. In §§5–7, we discuss how we obtain integer solutions by a price-and-branch heuristic, comment on implementation details of our column-generation code, and present computational experience drawn from practical instances. In fact, besides solving comparably large pickup and delivery problems, we demonstrate the ability of our strategy to obtain practically satisfactory solutions.

1. Engine Scheduling Problem

Because we aim at an effortless consideration of future extensions, our presentation in this section slightly generalizes the actual practical situation. In particular, our notion of *transportation* embodies more than just movement of rail cars, but the entire variety of tasks we only alluded to in the Introduction. For a planning horizon of about two hours, we are given a set $\mathcal{R}$ of n transportation requests to be served by a set $\mathcal{E}$ of m available engines. The node set $\mathcal{N}$ of a directed graph $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ contains origin and destination tracks r^+ and r^-, respectively, for each $r \in \mathcal{R}$ as well as start and end of service locations e^+ and e^-, respectively, for each engine $e \in \mathcal{E}$. Arcs $(i, j) \in \mathcal{A}$ are weighted with a travel time $t_{ij} \geq 0$ and cost $c_{ij} \geq 0$, both possibly infinity if the link does not exist. We mention that nodes correspond to logical locations rather than to physical ones. Each $i \in \mathcal{N}$ is attributed the size ℓ_i of the load to be picked up ($\ell_i > 0$) or delivered ($\ell_i < 0$), and a service time $s_i \geq 0$ to accommodate, for example, preparatory work at a track. Service must start within a time window (a_i, b_i). Arrival in i earlier than a_i, for example, waiting, is allowed; $\ell_i = 0$ is possible, but an engine's tractive effort L_e, $e \in \mathcal{E}$, must never be exceeded. Various objectives involving time and/or cost are conceivable, and two examples are given later in the paper.

Characteristic to multiple-vehicle pickup and delivery problems with time windows (m-PDPTW) (see the comprehensive survey by Savelsbergh and Sol 1995) are the assignment of engines to requests, decisions about the visiting time for each location, the precedence relation between origin and destination tracks, which must both be visited by the same engine (pairing), and the engine capacity constraint. However, contrary to published work, we deal with railroad traffic, and face the necessity that the sequence of consecutively visited locations must be chosen so as to avoid unnecessary switching operations: Railroad yard managers insist that every admissible schedule for an engine must have a very simple structure. That is, the corresponding path in $\mathcal{G}$ is constructed by sequentially visiting node sequences taken from the set

$$\mathcal{P} \subseteq \bigcup_{i \in \mathcal{R}} \{i^+, i^-\} \cup \bigcup_{i \neq j \in \mathcal{R}} \big\{\{i^+, j^+, i^-, j^-\} \cup \{i^+, j^+, j^-, i^-\}\big\}. \tag{1}$$

That is, the elements of $\mathcal{P}$, which we term *patterns*, only allow for *direct delivery*, and the simultaneous service of two requests precisely in the manner stated. We call the latter *overlapping* and *embedding*, for the temporal relationship of involved requests (see Figure 1). It is important to see that these patterns are part of the problem, not its simplification. They reduce shunting movements and are easy to perform for engine drivers. Note, that (1) only constrains the admissible sequence of locations, not the actual visiting times or the assigned engine. In that sense, the concept differs from those used, for example, in the airline crew pairing literature. We require that $\mathcal{P}$ only contain patterns that respect the engines' tractive efforts L_e. Patterns must not be a priori time window infeasible. Note that pairing and precedence of pickup and delivery locations are observed by definition. Given $\mathcal{P}$, let us refer to paths $R_e \subseteq \mathcal{N}$,

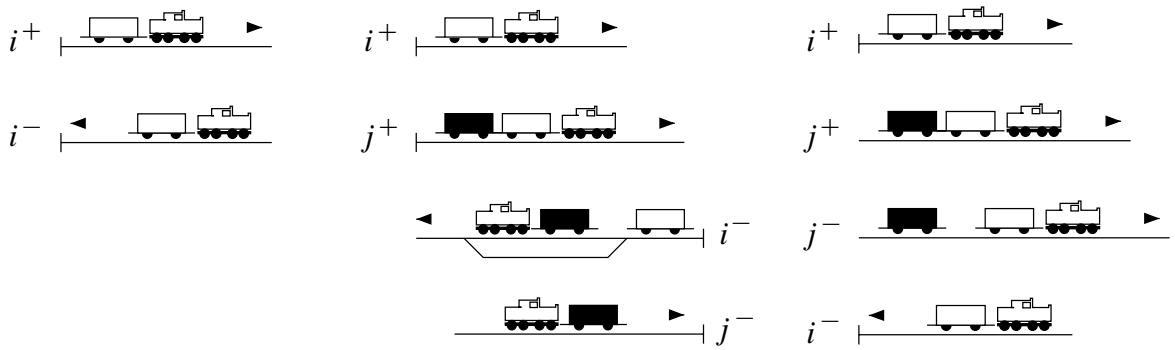

Figure 1 Typical Situations for Direct Delivery, Overlapping, and Embedding Requests.

$e \in \mathscr{E}$, from e^+ to e^-, constructed as sequences of (node disjoint) $P \in \mathscr{P}$ as $\mathscr{P}$ *concatenations*. A feasible $\mathscr{P}$ concatenation for engine $e \in \mathscr{E}$ is one which does not violate any time windows and visits only requests admissible on engine e. For an in-depth discussion of this concept and further applications see Lübbecke (2001a). We refer to Lübbecke (2001b) and Lübbecke and Zimmermann (2002) for a much more detailed discussion of the practical situation.

Given $\mathscr{E}$ and $\mathscr{R}$, a feasible solution to the *engine scheduling problem* (ESP) is a set $\{R_e\}_{e\in\mathscr{E}}$ of feasible $\mathscr{P}$ concatenations such that their disjoint union visits all nodes. That is, $\bigcup_{e\in\mathscr{E}} R_e = \mathscr{N}$. A potential objective function is to minimize the sum of arc weights along all concatenations, for instance, their lengths. It is easy to see that the multiple-traveling-salesman problem with time windows reduces to ESP, which is therefore $\mathscr{NP}$ complete in the strong sense (Lübbecke 2001b). This holds even for finding a feasible solution only.

2. Models for Engine Scheduling

2.1. Mixed Integer Formulation

In order to gather insight into the problem's structure, we formulate a mixed integer program, related to the *m*-PDPTW (Dumas et al. 1991, Sol 1994). It is modified for our special situation. Some additional notation and definitions are necessary. Given a pattern $P = \{i_1, i_2, \dots, i_k\} \in \mathscr{P}$, $\mathscr{A}(P) \subseteq \mathscr{A}$ denotes the arcs within P, i.e., $\mathscr{A}(P) = \{(i_1, i_2), (i_2, i_3), \dots, (i_{k-1}, i_k)\}$. Moreover, $o(P) = i_1$ and $d(P) = i_k$ are used for the *origin* and the *destination*, respectively, of the pattern P. In addition, $o(\{e^-\}) := e^-$ and $d(\{e^+\}) := e^+$. For any $e \in \mathscr{E}$ we define the service times $s_{e^+} = s_{e^-} = 0$. For a pattern $P \in \mathscr{P}$, $\boldsymbol{\delta}_P$ denotes its request incidence vector, the rth component δ_{rP}, $r \in \mathscr{R}$, which equals 1 if $r^+, r^- \in P$, and 0 otherwise. Finally, we denote the respective subset of patterns for which engine $e \in \mathscr{E}$ is admissible by $\mathscr{P}_e \subseteq \mathscr{P}$. We now introduce the variables:

- z_P^e 1 if pattern $P \in \mathscr{P}_e$ is assigned to engine $e \in \mathscr{E}$, 0 otherwise.
- $x_{P_1P_2}^e$ 1 if pattern $P_2 \in \mathscr{P}_e$ is immediately served after $P_1 \in \mathscr{P}_e$ on $e \in \mathscr{E}$, 0 otherwise.
- T_i (nonnegative) arrival time in location $i \in \mathscr{N}$ of the visiting engine.

Note that for the arrival times we need not specify *which* engine arrives at a particular location because this information is already covered by the z variables. Table 1 shows the entire formulation (2) through (11), denoted by (ESPMIP).

The given objective function (2) serves as a convenient example only. We aim at an earliest possible completion time for each engine. We ensure by (3) that the selection of patterns indeed partitions $\mathscr{R}$. An engine must visit and leave precisely those patterns assigned to it, which is guaranteed by (4). By constraints (5) and (6) each $\mathscr{P}$ concatenation starts and ends in the appropriate locations for the relevant engine. Note that this formulation allows that engines stay idle. The next two constraints take care of setting the visiting times of each particular location, (7) within a selected pattern and (8) between selected patterns. Time windows for the start-of-service time are respected due to (9). Finally, (10) and (11) define

Table 1 The Mixed Integer Formulation (ESPMIP) for the Engine Scheduling Problem

minimize	$\sum_{e\in\mathscr{E}} T_{e^-}$		(2)
subject to	$\sum_{e\in\mathscr{E}}\sum_{P\in\mathscr{P}_e} \delta_{rP} z^e_P = 1$	$\forall r \in \mathscr{R}$	(3)
	$\sum_{P\neq P_1\in\mathscr{P}_e\cup\{e^+\}} x^e_{P_1P} = \sum_{P\neq P_2\in\mathscr{P}_e\cup\{e^-\}} x^e_{PP_2} = z^e_P$	$\forall e \in \mathscr{E}, P \in \mathscr{P}_e$	(4)
	$\sum_{P\in\mathscr{P}_e\cup\{e^-\}} x^e_{\{e^+\}P} = 1$	$\forall e \in \mathscr{E}$	(5)
	$\sum_{P\in\mathscr{P}_e\cup\{e^+\}} x^e_{P\{e^-\}} = 1$	$\forall e \in \mathscr{E}$	(6)
	$z^e_P = 1 \Rightarrow T_i + s_i + t_{ij} \le T_j$	$\forall e \in \mathscr{E}, P \in \mathscr{P}_e, (i,j) \in \mathscr{A}(P)$	(7)
	$x^e_{P_1P_2} = 1 \Rightarrow T_{d(P_1)} + s_{d(P_1)} + t_{d(P_1)o(P_2)} \le T_{o(P_2)}$	$\forall e \in \mathscr{E}, P_1 \neq P_2 \in \mathscr{P}_e \cup \bigcup_{e\in\mathscr{E}}\{e^+, e^-\}$	(8)
	$a_i \le T_i \le b_i$	$\forall i \in \mathscr{N}$	(9)
	$z^e_P \in \{0,1\}$	$\forall e \in \mathscr{E}, P \in \mathscr{P}_e$	(10)
	$x^e_{P_1P_2} \in \{0,1\}$	$\forall e \in \mathscr{E}, P_1 \neq P_2 \in \mathscr{P}_e \cup \bigcup_{e\in\mathscr{E}}\{e^+, e^-\}$	(11)

the domains of the binary variables. For $i \in \mathscr{N}$ we assume $0 \le a_i$, therefore $0 \le T_i$.

Although we give a different motivation, we formally arrive at the model presented in the next subsection by means of a Dantzig-Wolfe decomposition. The nonlinear constraints (7) and (8), which then appear in the subproblem, do not constitute a complication because it is not solved as a linear program—the lower bound provided is poor (Lübbecke 2001b)—but by dynamic programming. We refer to Desaulniers et al. (1998) for a unified and much more detailed presentation of solution techniques.

2.2. Set Partitioning Formulation

Every feasible solution to the ESP canonically partitions the set of requests. Because every request must be covered by exactly one $\mathscr{P}$ concatenation, it is a natural idea to base decisions on entire concatenations, resulting in a set partitioning formulation. Denote by Ω_e the set of *all* subsets $R \subseteq \mathscr{R}$ with the property that a feasible $\mathscr{P}$ concatenation for engine $e \in \mathscr{E}$ visiting exactly all $r \in R$ exists. The elements of Ω_e will be called the *admissible request sets* for engine $e \in \mathscr{E}$. Associate with each $R \in \Omega_e$ an incidence vector $\boldsymbol{\delta}_R$ whose rth component δ_{rR} equals 1 if $r \in R$ and 0 otherwise. Usually, given $R \in \mathscr{R}$, there exist many different feasible concatenations visiting exactly all $r \in R$. However, we do not lose any generality when we assign to each $\boldsymbol{\delta}_R$ the smallest possible cost coefficient c^e_R with respect to the given cost evaluation of concatenations for engine $e \in \mathscr{E}$. Identical columns with larger cost will not be part of an optimal selection. Thus, we obtain a way to reconstruct a concatenation from its incidence vector. Whether a particular $R \subseteq \Omega_e$ is selected or not is represented by a binary variable λ^e_R. To summarize, this model reads as follows:

$$
\begin{aligned}
\text{Min} \quad & \sum_{e\in\mathscr{E}}\sum_{R\in\Omega_e} c^e_R \lambda^e_R & & \\
\text{subject to} \quad & \sum_{e\in\mathscr{E}}\sum_{R\in\Omega_e} \delta_{rR}\lambda^e_R = 1, & r \in \mathscr{R} & \\
& \sum_{R\in\Omega_e} \lambda^e_R \le 1, & e \in \mathscr{E} & \qquad \text{(ESPSP)} \\
& \lambda^e_R \in \{0,1\}, & e \in \mathscr{E}, & \\
& & R \in \Omega_e. &
\end{aligned}
$$

In other words, choose for each engine $e \in \mathscr{E}$ at most one admissible request set $R \in \Omega_e$ such that the disjoint union of chosen sets precisely yields $\mathscr{R}$ at a minimum cost. This seemingly compact formulation has some serious drawbacks. At first, note that determining the cost coefficient of a particular variable itself is an $\mathscr{NP}$ complete combinatorial optimization problem because it amounts to solving a 1-ESP. Second, although the model has very few constraints, in general each Ω_e is of exponential cardinality, resulting in prohibitively many variables. Thus, even the solution of the LP relaxation in a straightforward manner is impracticable.

3. Restricted Master Program

The tremendous number of variables of (ESPSP) leads us to use the well established column generation methodology. (See, for example, the surveys by Barnhart et al. 1998, Desaulniers et al. 1998, Desrosiers et al. 1995, and Lübbecke 2001b). Given a subset $\Omega'_e \subseteq \Omega_e$ of admissible request sets for each engine $e \in \mathscr{E}$, the restricted master program is

$$\begin{aligned} \min \quad & \sum_{e\in\mathscr{E}} \sum_{R\in\Omega'_e} c^e_R \lambda^e_R \\ \text{subject to} \quad & \sum_{e\in\mathscr{E}} \sum_{R\in\Omega'_e} \delta_{rR} \lambda^e_R = 1, \quad r \in \mathscr{R} \\ & \sum_{R\in\Omega'_e} \lambda^e_R \le 1, \quad e \in \mathscr{E} \\ & \lambda^e_R \ge 0, \quad e \in \mathscr{E}, R \in \Omega'_e, \end{aligned} \tag{RMP}$$

where the upper bound of one on the variables is already implied by the partitioning constraints. Initially, $\Omega' := \bigcup_{e\in\mathscr{E}} \Omega'_e$ can be thought of being *any*, even an empty, collection of admissible request sets that permit a partition of $\mathscr{R}$. An initialization is similar in spirit to the first phase of the simplex method. Associated with a primal optimal solution to (RMP) is a dual optimal solution $(\boldsymbol{u}, v)^\mathsf{T}$. The dual variables u_r, $r \in \mathscr{R}$, relate to the partitioning equalities and are therefore not restricted in sign. The nonpositive dual variables v_e, $e \in \mathscr{E}$, correspond to the convexity constraints. From the reduced cost coefficients $c^e_R - \boldsymbol{u}^\mathsf{T} \cdot \boldsymbol{\delta}_R - v_e$, $e \in \mathscr{E}$, $R \in \Omega_e$ we obtain the pricing problem

$$\min \left\{ c^e_R - \sum_{r\in R} u_r - v_e \;\middle|\; e \in \mathscr{E}, R \in \Omega_e \right\}, \tag{12}$$

which has to be solved in each iteration of the column-generation process.

4. Pricing Problem

The pricing problem (12) can be decomposed into subproblems, viz. one for each engine $e \in \mathscr{E}$,

$$z^\star_e := \min \left\{ c^e_R - \sum_{r\in R} u_r - v_e \;\middle|\; R \in \Omega_e \right\}, \tag{13}$$

where the dual variable value v_e corresponding to $e \in \mathscr{E}$ constitutes an additive constant that does not interfere with the minimization process. The minimum $\min_{e\in\mathscr{E}} z^\star_e$ determines a column to be adjoined to the restricted master program. We call (13) the *engine scheduling pricing problem*, or ESPP for short. It consists of finding a shortest (request disjoint) $\mathscr{P}$ concatenation for one fixed engine with the additional cost of $-u_r \in \mathbb{R}$ incurred for each visited request $r \in \mathscr{R}$. We occasionally use the notion of ESPP *concatenation*. Note that in this constrained shortest path problem we are not required to visit *all* the requests, but only a (possibly empty) subset. By reduction from LONGEST PATH we establish strong $\mathscr{NP}$ completeness of this problem (Lübbecke 2001b).

4.1. A Label Correcting Algorithm for ESPP

Given a set $\mathscr{P}$ of patterns, let us construct from $\mathscr{G} = (\mathscr{N}, \mathscr{A})$ the associated *pattern graph* $\mathscr{G}_\mathscr{P} = (\mathscr{P}, \mathscr{A}_\mathscr{P})$ by $(P_1, P_2) \in \mathscr{A}_\mathscr{P}$ if and only if $P_1 \cap P_2 = \emptyset$. We can represent request disjoint $\mathscr{P}$ concatenations as simple paths in $\mathscr{G}_\mathscr{P}$ (the converse is not true), and we will implicitly work on this graph. Arc weights are redefined as $\bar{c}_{ij} := c_{ij} - u_r$ if $i = r^+$, and $\bar{c}_{ij} := c_{ij}$ otherwise, for all $(i, j) \in \mathscr{A}$, and transferred to the respective arcs in $\mathscr{G}_\mathscr{P}$. Because we solve a separate pricing problem for each engine $e \in \mathscr{E}$, the only relevant requests are those admissible on engine e. These are denoted by $\mathscr{R}_e \subseteq \mathscr{R}$.

An important concept in the design of combinatorial shortest path algorithms is to maintain a set of distance labels, one for each node. Starting at infinity and being improved in the iterative course of the algorithms, its value is at any time an upper bound on the length of a shortest path from the source e^+ to node i, equaling the respective optimal length upon termination (Ahuja et al. 1993). When we wish to keep track of both time and cost, however, one-dimensional labels do not suffice. (See Desrosiers et al. 1983.) Because two-dimensional labels can be ordered only partially, a list of (nondominated) labels must be maintained at each node.

With each delivery node i we associate *states* (R, i) and *labels* (T^k_i, C^k_i). These specify the kth ESPP concatenation that visits requests $R \subseteq \mathscr{R}_e$, and ends in node i. The time start-of-service time is T^k_i at a total reduced cost of C^k_i. The fundamental step of the algorithm is the treatment of a label (T^k_i, C^k_i), i.e., the

construction of all feasible extensions of the corresponding path, each by (usually) one additional arc emenating from i, resulting in new paths and the associated labels, respectively. Treating all the labels at a state is called the treatment of this state. Whenever new labels are generated at state (R, i), these may dominate others already present, thus offering possible improvement. Then, state (R, i) must be treated (again) to propagate this improvement. The process terminates as soon as no new labels are generated at any state. Paths are extended by several nodes at once, viz. patterns taken from $\mathcal{P}$. The procedure is similar to algorithms for the construction of (elementary) constrained shortest paths used, for example, in the context of airline scheduling or urban transit problems; see Gegen et al. (2000). In contrast to these problems, however, we do not have a fixed schedule, but a flexible one. That is, we must respect time windows.

At a state, we store labels in chronological order. When time window infeasibility is detected while we try to expand a label, no further labels at the current state need to be treated—none will be feasible either. States are treated in chronological order as well. Note that label $(T^1_{e^+}, C^1_{e^+})$ is initialized with $C^1_{e^+} = -v_e$, the dual variable value associated with engine e. Alternatively, a fixed cost incurred by using this engine may be added. We keep a list of states still to be treated. All lists of labels appearing in the course of the algorithm for the first time are initialized as empty sets. Request disjointness of paths needs to be checked when a label is treated. Note that we may use pattern families more complicated than (1). We refer to Lübbecke (2001b) for a more formal description of the algorithm. Upon termination, the output is the column-incidence vector $\boldsymbol{\delta}_{R^\star}$ at cost

$$C^{k^\star}_i + \sum_{r \in R^\star} u_r + v_e, \qquad \text{where}$$
$$(R^\star, k^\star) \in \operatorname*{arg\,min}_{(R,k)} \left\{ C^k_i \text{ at state } (R, i) \right\} \tag{14}$$

if $C^{k^\star}_i < 0$ and otherwise is the information that no more columns with negative reduced cost exist. In the event that the pricing problem is required to return multiple columns, cf. §6.1, we may evaluate (14) for each delivery location i. Heuristically, this leads to a certain *diversity* of the returned solutions. We do not discuss an efficient label management, and only refer to techniques introduced by Desrochers and Soumis (1988a, b) and Desrosiers et al. (1983), which are applicable. (See Lübbecke 2001b.)

4.2. Dual Variable-Based Label Elimination

Dominance among states is the classical means to reduce the solution space of a dynamic program. This depends on the implementation and requires a good organization of the label space. Moreover, we can only compare labels *already constructed*. Rather, it would be preferable to have an *anticipatory prevention* of unpromising labels. In branch-and-bound methods, a good performance essentially hinges on good bounds. These two well-known implicit enumeration strategies exhibit large similarities. (See, for example, Ibaraki 1987.) The common framework motivates us to investigate lower bounds in the context of reducing the number of labels in our dynamic program. To this end, we apply a technique introduced recently by Lübbecke (1999).

When solving ESPP, we are given a fixed engine $e \in \mathcal{E}$. With respect to (13), we implicitly explore Ω_e, the set of admissible request sets for engine e. During the search, denote by $\overline{\Omega}_e \subseteq \Omega_e$ the subset so far considered, i.e., $R \in \overline{\Omega}_e$, if and only if we have already generated labels with finite cost for some state $(R, \cdot)$. Furthermore, we denote by C^{inc} the cost of a currently cheapest label, referred to as the *incumbent*. Initially, $C^{\text{inc}} = \infty$. Let R^{inc} denote the associated admissible request set. Note that C^{inc} is the optimal value sought by the procedure when $\overline{\Omega}_e = \Omega_e$. Efforts to eliminate labels aim at precluding as large a subset of Ω_e as possible from being searched, while guaranteeing that an optimal concatenation *will* be identified. Because ESPP is a pricing problem, we are interested in *negative* values of C^{inc} only, and it would be helpful to know if the treatment of a given state (R, i) *can* eventually lead to a state with negative reduced cost labels at all. Even stronger is the question whether *any* future treatment of (successor states of) (R, i) exists that yields a better incumbent. A negative answer immediately enabled us to prune the search, excluding (R, i) from further treatment, and

therefore reducing the searched state and label space. Clearly, as soon as

$$LB(R) \geq \min\{0, C^{\text{inc}}\} \tag{15}$$

holds for a lower bound $LB(R)$ on the best possible reduced cost coefficient obtainable by treatment of any label associated with states $(R, \cdot)$, we have such a negative answer. Denote by $\mathcal{R}_e^+$ the subset of requests admissible on engine e with associated positive dual variable value, for example, $\mathcal{R}_e^+ := \{r \in \mathcal{R}_e \mid u_r > 0\}$. Let $R \subseteq \mathcal{R}_e$ be the admissible request set corresponding to a particular state under consideration. The only way to lessen the cost of the labels associated with this state is to expand the corresponding concatenations by requests in $\mathcal{R}_e^+$. Hence, a way of providing a lower bound to be used in (15) is

$$LB(R) := \min_{i \in R} C_i^{\hat{k}} - \sum_{r \in \mathcal{R}_e^+ \setminus R} u_r, \tag{16}$$

where the $\hat{k}$th label corresponds to the least cost label at state (R, i). In our computational results this criterion proves particularly effective, cf. Table 2.

4.3. Heuristics

An exact algorithm for ESPP is indispensable when a certificate of optimality is needed upon completion of the column-generation algorithm. However, especially in the beginning, a pricing algorithm may return *any* column with negative reduced cost coefficient. Heuristics are a way to do this fast. Here, we consider heuristic variants of our algorithm. That is, we waive optimality by restrictions on its solution space. These variants are separately presented in turn, but can be arbitrarily combined.

Premature Termination. An immediate modification is to return the column corresponding to the first, or more generally, to the kth constructed negative reduced cost label, with a given parameter $k \in \mathbb{N}$. This parameter controls the trade-off between solution quality and computation time. Beyond being very easy to implement, this idea is attractive for its implicitly *dynamic* behavior. Without further algorithmic efforts, the method finally delivers an optimal solution as well.

Table 2 Results for vps **Instances**

Instance	LP opt	IP obj	%gap	B&B	CPU Total	w/o (15)
vps10	1.000	1.0	**0.00**	0.01	0.05	0.10
vps11	1.000	1.0	**0.00**	0.00	0.05	0.24
vps12	1.000	1.0	**0.00**	0.00	0.05	0.25
vps13	1.000	1.0	**0.00**	0.00	0.06	0.31
vps14	9.000	9.0	**0.00**	0.00	0.05	0.24
vps15	8.000	8.0	**0.00**	0.01	0.10	0.65
vps16	8.000	8.0	**0.00**	0.03	0.12	0.55
vps17	8.500	9.0	**0.00**	0.07	0.20	1.29
vps18	8.500	9.0	**0.00**	0.04	0.14	1.30
vps19	8.000	8.0	**0.00**	0.05	0.23	1.61
∅	5.400	5.5	**0.00**	0.02	0.10	0.65
vps20	8.000	8.0	**0.00**	0.33	0.63	2.09
vps21	8.000	8.0	**0.00**	0.16	0.44	5.94
vps22	8.000	8.0	**0.00**	0.03	0.17	3.76
vps23	8.000	8.0	**0.00**	0.10	0.32	9.36
vps24	12.000	12.0	**0.00**	0.27	2.25	11.88
vps25	8.000	8.0	**0.00**	0.08	1.31	19.60
vps26	8.000	8.0	**0.00**	0.07	2.25	58.30
vps27	16.000	16.0	**0.00**	0.09	5.80	65.41
vps28	20.000	20.0	**0.00**	0.12	11.25	73.18
vps29	19.000	19.0	**0.00**	0.16	10.33	145.25
∅	11.500	11.5	**0.00**	0.14	3.48	39.47
vps30	19.000	19.0	**0.00**	0.20	15.32	388.20
vps31	18.000	18.0	**0.00**	0.31	26.50	512.95
vps32	16.750	17.0	**0.00**	0.84	25.25	1347.25
vps33	16.750	17.0	**0.00**	0.89	30.52	15224.63
vps34	16.750	17.0	**0.00**	1.31	96.29	54707.18
vps35	16.750	17.0	**0.00**	0.86	58.45	501309.84
vps36	15.000	15.0	**0.00**	0.88	124.10	†
vps37	22.500	24.0	4.34	4.41	1862.31	†
vps38	22.500	24.0	4.34	2.85	1321.49	†
vps39	22.500	27.0	17.39	16.32	2429.00	†
∅	18.650	19.5	2.61	2.89	598.92	

Note. Under heading "w/o (15)" we list the total computation time in CPU seconds, when the lower bound criterion (15) is *not* used. The sign † means that every reasonable time bound is exceeded, even for testing purposes. (Here: one CPU week.)

Cost Coefficient Improvement. For every heuristically generated column-incidence vector $\boldsymbol{\delta}_R$, the respective cost coefficient c_R^e is not necessarily minimal. This implies that the same column is potentially regenerated with smaller cost coefficient in a later column-generation iteration. When $|R|$ is not too large, say $|R| \leq 4$, it is advantageous to calculate the best possible c_R^e by determining an optimal ESPP concatenation

for the fixed request set R, for example, by a simple backtracking algorithm.

Pattern Graph Reduction. A heuristic reduction of the number of nodes and arcs (Desrosiers et al. 1986, Dumas et al. 1991) reduces the number of generated labels. Moreover, when only direct delivery patterns are considered, we reduce the number of nodes by an order of magnitude. This latter heuristic usually produces feasible solutions because it reflects the current planning practice. For our instances, the speed up is marginal, but solution quality drops. Similar to Crainic and Rousseau (1987), we may alternatively omit patterns involving $r_1, r_2 \in \mathcal{R}_e$ with $\sum_{(i,j)\in\mathcal{A}(P)} c_{ij} - u_{r_1} - u_{r_2} \geq 0$.

Alternative Treatment of States. Instead of considering states (R, i) in chronological order, a possible treatment is in nondecreasing order of out-degree of node i in the pattern graph. The rationale behind this modification is that label elimination criteria are most effective when applied early in the search, thus pruning large portions of the tree in Figure 2. This is the more likely the farther from the root the *fanout* appears (Reingold et al. 1977). Alternatively, we may order states (R, i) according to nonincreasing value of the dual variable associated with the request with delivery location i. The motivation here is to construct good incumbents early.

A different strategy aims at the heuristic elimination of all states with $|R| \leq \rho$, where $\rho \in \mathbb{N}$ is a preset parameter. For instance, $\rho = \lceil |\mathcal{R}|/|\mathcal{E}| \rceil$ is a reasonable choice, when the request load on the engines is supposed to be almost balanced, and in our case leads to good results.

5. Price-and-Branch

In general, solving an LP relaxation by column generation need not yield an integral solution. Even worse, the variables generated at LP optimality need not allow for an integer *feasible* solution at all. Our computational experiments indicate that our restricted master programs happen to already be integer feasible, and solution quality is fully acceptable. The column-generation process is not invoked at any node other than the root node of the branch-and-bound tree. This policy may be termed *price-and-branch* by analogy with the folklore notion *cut-and-branch*, where

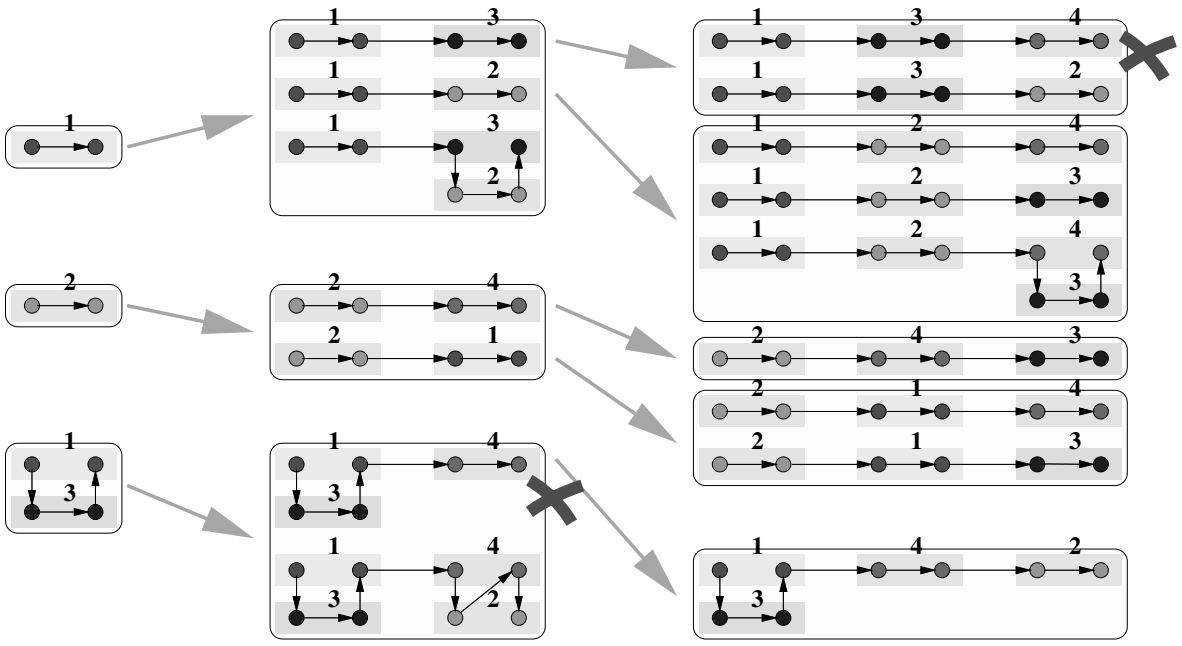

Figure 2 Another Point of View of Extending Concatenations

Note. The marked concatenations are stored in the same state $(\{1, 3, 4\}, 4^-)$, thus allowing for an efficient elimination of a possibly dominated label.

valid inequalities are adjoined to the LP at the root node only.

Here again, we come across the phenomenon that reality is not as bad as suggested by computational complexity or, in other words, in practice the theoretically worst case rarely occurs. This was observed in many other practical settings as well where artificial data were compared to "real-world" data. Although not being a mathematically satisfactory explanation, this is an encouragement to attack practical problems, even in the presence of negative complexity results. Of course, in a failsafe industrial implementation, one could safeguard against infeasibilities by including a branch-and-price code. However, this is beyond the scope of this paper.

6. Implementation Details

In this section we describe in detail the implementation of our price-and-branch heuristic. Efficiency is of primary importance when practical problem sizes are to be addressed. It is our experience, in line with that of other researchers, that generic column generation offers a vast degree of freedom from an implementation standpoint. Most recently, a compendious collection of implementation and acceleration techniques was presented by Desaulniers et al. (2001). Although sketchy, we believe that presently their review can hardly be outmatched in terms of experience and scope. It should be definitively considered before an implementation.

The objective function we actually implemented is the practically relevant minimization of the total dead heading and waiting time. Then, the cost for a concatenation cannot simply be calculated by straightforward addition of arc weights.

6.1. Subproblem Solution and Column Management

The pricing subproblem is the most frequently executed essential component of a column-generation code. Each call should therefore be as effective as possible. Even though this statement seems obvious, it deserves some recognition. Recall that it is neither mandatory to add a most profitable column to the restricted master program, nor are we restricted to adding only one column at a time. Not only the heuristics we discussed, but also our exact label correcting algorithm, exhibit a large degree of freedom in their actual realization and combination. It is our experience that the latter aspect outweighs the still valid importance of an efficient implementation of the respective components. The global guideline is to call our *exact* pricing algorithm as seldom as possible. Other authors share this point of view, but in our case this is all the more important inasmuch as we have to solve a *node disjoint* constrained shortest path problem, a requirement often absent in similar situations.

6.1.1. Column Pool. We stop as soon as the kth negative reduced cost column is found, where $k \in \mathbb{N}$ is parameter controlled. Still, negative cost columns are produced that are *not* added to the restricted master program. The efforts for generating them appear to be in vain. Instead, we would store all columns with reduced cost smaller than a threshold to a so-called *column pool*. This concept has been successfully applied in column-generation codes. (See, for example, Sol 1994.) Before any other construction method is invoked, we check whether the pool contains columns that price out negatively. It may also be helpful to add columns to the restricted master with a small positive reduced cost. These may become active in later iterations. However, such columns are only added when, in the same iteration, at least one negative reduced cost column is added from the column pool. Otherwise, cycling could occur.

The pool is linearly searched, performing for each entry (a) an update of reduced cost according to the current dual variables, (b) the addition of the column to the restricted master (and deletion from the pool), if applicable, and (c) a deletion of the column from the pool if it is too expensive. All thresholds are parameter controlled. We neither limit the pool capacity nor the number of columns to be added to the pool or to the restricted master per iteration. The selection of partial solutions is proposed by Savelsbergh and Sol (1998), for example, a set of columns that cover each request once at most and originate from different engines. In our experiments this is unnecessary in terms of integer feasibility. However, we try to keep a pool of high quality. That is, when a column is pushed

to the pool we check whether its reduced cost can be improved (by a backtracking method).

6.1.2. Greedy Dual Variable/Minimal Time Window Slack Heuristic. When the pool contains no more suitable columns, we invoke a very simple heuristic that greedily assigns direct delivery patterns to the engine in question as long as it is time feasible. Each decision should provide a good decrease of the tentative column's reduced cost coefficient but maintain time window feasibility as long as possible. That is, a compromise is sought between a large positive dual variable value and the least detriment of the degree of freedom for the remaining decisions. Therefore, we choose the next request to be assigned as a maximizer of the following ranking. Divide the dual variable value associated to a request by the maximal number of minutes the engine could be delayed while still completing the request in time. In other words, a critically small time window slack amplifies the dual variable weight of a request. See Lübbecke (2001b) for a more formal description.

6.1.3. Premature Termination of the Exact Label Correcting Algorithm. When the above greedy heuristic fails to provide a negative reduced cost column we invoke our prematurely terminating label correcting algorithm. Note that this heuristic *will* return a promising column if one exists. We proceed in the treelike manner as indicated in Figure 2. When *no* column is found for *any* engine, we turn off all still active heuristics if any, thus finally calling an exact pricing algorithm to add final improving columns and to prove optimality.

6.1.4. Multiple Pricing. We cyclically price the engines. During the first master iterations the heuristics usually deliver a good column. Later on, more and more often the label correcting algorithm is called which simultaneously refills the column pool. After each such pricing, the restricted master is reoptimized. The motivation for not pricing *all* engines before reoptimization is to provide the respective pricing problems with the most recent dual information, which of course reflects results from prior pricing problems. This is simply a heuristic to avoid generation of too many similar columns.

7. Computational Experience

All experiments were performed on a 700 MHz Pentium III PC with 1 GByte core memory running Linux 2.2. We use the `CPLEX` 7.0 callable library to solve all linear and integer programs. Compilation with the GNU C compiler `gcc` is invoked with `-O3` optimization.

7.1. Available Data

7.1.1. VPS. *Verkehrsbetriebe Peine-Salzgitter GmbH, Germany*, provided us with raw data that represent 100 actual requests from a morning shift of 8.5 hours. All requests originate from operating two blast furnaces at a steel mill, including some additional requests, such as engineer's breaks. Two engines are primarily dedicated to slag transport and four engines to the transportation of molten iron. Only 21 tracks are involved, and distances are relatively short. Creating instance `vps100` from the raw data necessitated considerable manual modifications that have been done with approval by railroad officers. In particular, no time windows are given but are to be extracted from handwritten data sheets. From the chronologically sorted instance `vps100` we derive a new instance `vps`*nn* by considering the first *nn* requests only. The practical background requires a large fraction of requests to be directly delivered because much surveillance activity is necessary for safety reasons. A particularity of the data set is that there are many requests compared to the length of the planning horizon.

7.1.2. EKO. The raw data furnished by *EKO Trans GmbH, Eisenhüttenstadt, Germany*, comes from a steel mill again. This railroad is a smaller one and operates 10 engines altogether, each of which is assigned a certain region and/or tasks. In total, we have 820 requests that are generated from a log file of rail car movements on the entire industrial plant during six consecutive days. The requests cover 273 tracks spread all over the plant. No time window information is available for this data set. Therefore, we produce time windows of a parameter controlled width, centered at the actual start-of-service time. This yields instances `eko820a`, `b`, and `c` with time window widths

of 200, 250, and 300 minutes, respectively. For our calculations we build groups of 25 requests, such that the time windows span about one half of the respective planning horizon. For many origin destination pairs, the distance is estimated. Admittedly, the result cannot be regarded as real-life data.

We also generate a set of extremely difficult instances, where time windows are fairly wide and most distances are *very* short. Then, *many* request sets of almost identical cost are admissible. This results in many columns being favorably added to the restricted master program, only a tiny fraction of which are actually needed. Instances `eko80xa`, `b`, and `c` each contain requests 1–80 with time window widths of 100, 200, and 300 minutes, respectively.

7.1.3. Sol. We are kindly provided with several randomly generated m-PDPTW instances from the literature (Savelsbergh and Sol 1998, Sol 1994). These are used as raw data for our own experiments. Eight problem classes of ten Euclidean instances each are given. We have homogeneous fleets, and $|\mathcal{R}|/2$ vehicles are available and admissible for each request. Classes `A`, `B`, and `DAR` have time windows of one hour length, and classes `C` and `D` of two hours length. Time windows fulfill $a_{r^-} - a_{r^+} = t_{r^+r^-}$ for all $r \in \mathcal{R}$. The last time window closes after about 11 hours; however, time windows that exceed the planning horizon of 600 minutes are truncated. Vehicle capacity is restricted to accommodate at most three (classes `A`, `C`), four (classes `B`, `D`), or five requests (classes `DAR`), respectively. We do not truncate time windows; therefore we obtain longer planning horizons. From the time window and capacity data we derive for our instances the respective sets of admissible patterns in $\mathcal{P}$. All other information remains intact.

7.2. Evaluation of Results

Some observations are generally valid across all instances. All final restricted master programs are integer feasible when default options are used. Solution quality is practically satisfactory. It must be mentioned that seemingly significant optimality gaps translate to only *minutes* from optimum in practice. We prematurely terminate our exact pricing when the 100th negative column is found. This enlarges the amount of pooled (and potentially added) columns, which in turn may allow for better integer solutions. Seldom are more than 5,000 columns generated, usually significantly less. With increasing planning horizons and wider time windows, problems become much harder to solve. In comparison, small amounts of time are spent in the final branch-and-bound phase. In what regards numerical stability, we remark that although all cost coefficients are integers, intermediate calculations involving dual variables do have results in the reals. We observe tiny deviations of about 10^{-15} from integral values, and round to the nearest integer. The column headings in the tables displaying our computational results have the following meanings:

Instance	name of the problem instance.
LP opt	optimal objective function value of the LP relaxation.
IP obj	best feasible integer solution value found.
%gap	relative optimality gap in percent.
B&B	CPU time in seconds spent in the branch-and-bound phase.
CPU total	total computation time in CPU seconds.

7.2.1. VPS. The `vps` data set is of special relevance to us because it reflects a typical practical situation. Table 2 demonstrates the excellent quality of the lower bound provided by (ESPSP), and thus the robustness of our price-and-branch approach. We obtain integer optimal solutions for a considerable number of requests within a few minutes. This certainly allows for interactivity with the dispatcher. Due to small absolute cost coefficients, we inhibit the generation of labels with cost larger than 30.

We also use these calculations for an evaluation of our lower bound label elimination criterion (15). For growing problem size, we obtain speedups of orders of magnitude. Using the same amount of time, we are enabled to handle additional requests—with considerably smaller memory requirement. The criterion is especially effective in the end of the column-generation process when it is particularly difficult to find negative columns. Further observations are that from a computational standpoint it is entirely

impractical to always run our exact label correcting algorithm until the end; starting with only an artificial basis is competitive to using a greedy heuristic; however, the penalty cost for artificial variables should be carefully chosen. Column *elimination* from the restricted master program is not favorable, especially because this reduces our chances to obtain integer solutions, and only minor progress is made in terms of the primal objective function, per master iteration.

7.2.2. EKO. We evaluate the limitations of our implementation with respect to time window width, cf. Table 3. In practice, we usually have a mix of requests that become available late compared to their required completion time, and requests that have extremely large time windows. Each group of 25 requests represents approximately one half of a shift. When time windows get wider (and all other data stays the same), the problems get significantly harder to solve. Additional comparatively short distances render our approach computationally impractical. It must be pointed out, however, that these results are obtained with default options active. For the most difficult group of requests computationally, 676–700 (time window width 300 minutes) we produce an integer solution with objective value of 356.0 (LP optimum is 243.0) in 1,000 CPU seconds, when the generation of labels is heuristically limited. With sufficient knowledge about the practical situation, such a limitation can always be successfully applied.

Table 3 **Results for Instances** `eko820a`, `b`, **and** `c`

Requests	Width	%	LP opt	IP obj	%gap	B&B	CPU Total
1–100	200	47	524.146	637.750	16.63	2.61	53
	250	49	414.958	431.000	5.29	1.77	690
	300	54	358.930	375.500	5.93	2.72	5364
101–200	200	42	448.794	471.250	4.73	1.18	29
	250	47	391.502	411.750	6.13	1.28	406
	300	51	363.296	406.750	12.60	6.04	4749
201–300	200	40	630.983	733.000	11.50	1.24	15
	250	45	525.646	560.750	10.34	3.10	144
	300	49	477.812	563.250	13.33	3.12	1892
301–400	200	43	512.180	551.500	7.75	0.44	54
	250	49	407.790	441.500	7.62	1.50	962
	300	53	370.219	387.750	4.17	1.02	11378
401–500	200	45	641.833	804.000	19.18	0.55	4
	250	50	429.229	499.500	14.34	3.47	11
	300	55	366.050	441.250	17.63	3.66	50
501–600	200	39	473.078	500.250	6.17	0.99	674
	250	45	411.723	433.000	6.38	4.20	2241
	300	49	357.389	379.750	8.77	1.55	19176
601–700	200	48	387.087	400.250	3.90	0.49	1760
	250	54	327.878	358.250	9.52	5.36	30590
	300	59	274.975	283.250	2.96	16.68	226231
701–800	200	39	436.679	484.500	9.09	1.22	30
	250	45	352.609	413.250	14.06	7.05	115
	300	49	306.879	463.750	36.00	15.44	338

Note. Thirty-two groups of twenty-five requests each are formed. We report the average figures for four groups, respectively, i.e., 100 consecutive requests. Under headings "width" and "%" we list the respective time window width in absolute value and relative to the planning horizon, respectively. Using a heuristic, these computation times can be dramatically sped up.

When there are almost no preferences given with respect to when a request should be served, combinatorial explosion strikes. Our algorithm is not able to handle too large a fraction of such requests as can be seen from Table 4. It appears that the difficulty with the hard instances `eko80x` lies in the enormous amount of feasible solutions. This can be deduced from a comparison of the number of generated columns and the number of columns actually needed; the number of simplex iterations serves as indicator here. Heuristically, as before, for the hardest instance we obtain an integer solution with 13% gap in only 400 CPU seconds in the column-generation phase.

7.2.3. Sol. In Tables 5 and 6 we list results for the data set taken from Sol (1994). On the general PDPTW instances, we impose the additional constraint of allowing $\mathcal{P}$ concatenations only. Interestingly enough, the resulting instances are feasible even for the originally shorter planning horizon of 600 minutes. Moreover, for the original instances we know the optimal total route durations (Sol 1994). The total route duration deduced from our integer solutions are, on average, at most 20% longer. In a sense, the original situation is restricted in such a way that using $\mathcal{P}$ concatenations is quite competitive, and solutions are quickly computed.

It must be stated explicitly that our objective function also differs from Sol's, in that we do not seek a solution with a minimal number of used vehicles because this is not favored by railroad management (yet). Still, remarkably, often the optimal number also

Table 4 Results for Hard Instances eko80x

Requests	Width	%	LP opt	IP obj	%gap	#columns	#simplex	B&B	CPU Total
1–20	100	19	526.000	526.000	**0.00**	1873	669	0.42	4
	200	32	483.000	483.000	**0.00**	1838	834	0.47	30
	300	41	458.000	458.000	**0.00**	1586	680	0.33	54
21–40	100	39	76.000	76.000	**0.00**	3612	2051	3.70	2715
	200	56	71.000	71.000	**0.00**	5968	2569	0.58	15056
	300	66	71.000	71.000	**0.00**	138559	1752	31.63	12271
41–60	100	34	87.000	87.000	**0.00**	2714	1880	2.26	1981
	200	51	87.000	87.000	**0.00**	61677	1963	10.93	18815
	300	61	87.000	87.000	**0.00**	83799	2173	16.59	19441
61–80	100	48	100.000	100.000	**0.00**	486097	4168	54.64	66658
	200	65	100.000	†		1343598	5813	199.84	274694
	300	73		†					

Note. We report results for groups of only 20 requests. Under headings "#columns" and "#simplex," respectively, we report the total number of columns adjoined to the RMP and the total number of simplex iterations needed to reoptimize the RMP. We are unable to obtain solutions for the last group, when time windows get wider, cf. the sign †. For these two problems the pattern graph has a density of 95%. However, for the last instance, premature termination of the column-generation phase after 400 CPU seconds leads to an integer solution with objective function value of 113!

is used in our solutions, and we seldom only use one vehicle in excess. We must observe, however, that our engines are allowed to exceed the original planning horizon. Hence, it happens, that the solution we calculate is better than the optimum stated by Sol (1994). For the DAR50 instances, cf. Table 6, we also use fewer vehicles than are optimal in the original situation. This is again a consequence of our enlarged planning horizon. With larger time windows *and* larger vehicle capacity (class D) solution times slightly increase and solution quality drops.

8. Remarks

Unfortunately, we are not able to compare our objective function values with those obtained in the real-life setting because engine assignments or actual service completion times are not recorded. Time window information is especially lacking. This is also due to the fact that our definition of a transportation request is broader and more flexible than the one used in practice. Although they appear to be small, the instances we are able to solve correspond to a planning horizon of more than two hours. According to railroad officers it is hardly sensible to plan further ahead because of the increasing uncertainty of future requests.

Our computational bottleneck is the exact pricing algorithm, and the practical implementation of the method hinges on a reasonably small cardinality of $\mathcal{P}$. The use of more elaborate data structures for the label management, and the further development of heuristics for reducing the underlying network would certainly bring further speedup. This is not part of our study. Still, we are able to state computational feasibility of our approach; everyday computability is to be established in a next step. We refer to Lübbecke and Zimmermann (2002) for supplementary reading on how to implement our work in practice. We would like to remark that the ESP's generic structure underlies several other logistics problems, see Lübbecke (2001a).

We avoided the additional effort of implementing a branch-and-price algorithm. This is well justified by our always obtaining (even optimal) integer solutions. From a practical, as well as from a theoretical perspective, it would be interesting to evaluate stochastic influences of engine availability and appearance of requests. There is also practically no knowledge about the so-called online version of our problem—certainly a research avenue for the years to come.

Table 5 Results for Small Instances by Sol (1994)

Instance	LP opt	IP obj	%gap	#eng/#opt	CPU Total	B&B
A30_1	1786.875	1836.000	2.74	**10/10**	2.00	1.04
A30_2	1981.000	1981.000	**0.00**	**11/11**	0.87	0.03
A30_3	3226.750	3409.000	5.63	14/13	5.41	4.88
A30_4	1794.000	1819.000	1.39	**11/11**	1.06	0.49
A30_5	2025.000	2072.000	2.32	**11/11**	1.45	0.88
A30_6	1837.000	1878.000	2.23	**11/11**	0.73	0.22
A30_7	1727.500	1734.000	0.34	**10/10**	0.77	0.14
A30_8	2350.000	2350.000	**0.00**	**11/11**	0.64	0.03
A30_9	2501.000	2501.000	**0.00**	**11/11**	0.68	0.11
A30_10	2594.000	2594.000	**0.00**	**11/11**	0.70	0.12
∅	2182.312	2217.400	1.46	11.1/11.0	1.43	0.79
B30_1	3024.000	3024.000	**0.00**	**12/12**	0.68	0.11
B30_2	1849.000	1891.000	2.27	**9/9**	1.49	0.51
B30_3	2503.000	2503.000	**0.00**	**12/12**	0.62	0.03
B30_4	2094.000	2094.000	**0.00**	**10/10**	1.06	0.05
B30_5	2072.500	2082.000	0.43	**9/9**	0.91	0.12
B30_6	2100.000	2100.000	**0.00**	**10/10**	0.63	0.03
B30_7	1980.200	2018.000	1.86	**10/10**	0.97	0.21
B30_8	2978.000	2978.000	**0.00**	**13/13**	0.44	0.05
B30_9	2968.000	2968.000	**0.00**	**14/14**	0.60	0.02
B30_10	2264.000	2264.000	**0.00**	**11/11**	0.89	0.15
∅	2383.270	2392.200	0.45	11.0/11.0	0.82	0.12
C30_1	1490.000	1490.000	**0.00**	**8/8**	1.08	0.03
C30_2	1528.000	1528.000	**0.00**	**8/8**	1.22	0.02
C30_3	1865.000	1888.000	1.23	**8/8**	3.33	0.40
C30_4	1215.100	1276.000	4.93	8/7	3.43	1.24
C30_5	1648.000	1648.000	**0.00**	**8/8**	1.60	0.05
C30_6	1985.000	2083.000	4.93	**9/9**	3.41	2.25
C30_7	1663.667	1675.000	0.66	**8/8**	1.81	0.17
C30_8	1830.250	1866.000	1.91	**9/9**	2.62	1.30
C30_9	1337.391	1414.000	5.68	**8/8**	2.60	1.17
C30_10	1676.833	1692.000	0.89	**8/8**	3.05	0.91
∅	1623.924	1656.000	2.02	8.2/8.1	2.41	0.75
D30_1	1539.000	1539.000	**0.00**	**9/9**	2.04	0.13
D30_2	974.000	974.000	**0.00**	**7/7**	7.38	0.05
D30_3	1357.500	1393.000	2.57	9/8	2.25	0.78
D30_4	1426.659	1494.000	4.69	9/8	2.53	1.00
D30_5	1359.452	1405.000	3.30	**8/8**	2.16	0.75
D30_6	1615.000	1681.000	4.08	**8/8**	3.71	2.53
D30_7	1109.288	1149.000	3.51	9/8	3.22	1.41
D30_8	1614.714	1687.000	4.45	8/7	2.13	0.54
D30_9	1340.636	1434.000	6.93	**8/8**	8.65	6.65
D30_10	1582.200	1719.000	8.59	8/7	18.60	17.12
∅	1391.844	1447.500	3.81	8.3/7.8	5.26	3.09

Note. Under heading "#eng/#opt" we display the number of engines we use in our solutions versus the respective optimal number.

Table 6 Results for Larger and Less Restricted Instances by Sol (1994)

Instance	LP opt	IP obj	%gap	#eng/#opt	CPU Total	B&B
A50_1	2993.000	2993.000	**0.00**	**17/17**	7.82	0.49
A50_2	3767.333	3946.000	4.72	**17/17**	65.32	60.46
A50_3	3367.000	3367.000	**0.00**	**16/16**	5.08	0.18
A50_4	2801.000	2801.000	**0.00**	**14/14**	7.52	0.06
A50_5	2554.000	2554.000	**0.00**	**20/20**	3.69	0.07
A50_6	2855.000	2855.000	**0.00**	**15/15**	9.19	1.47
A50_7	2892.857	2931.000	1.31	**17/17**	7.61	2.54
A50_8	2849.200	2872.000	0.77	**15/15**	9.71	2.08
A50_9	2690.667	2753.000	2.30	**14/14**	16.10	2.91
A50_10	2660.000	2660.000	**0.00**	**15/15**	4.72	0.07
∅	2943.005	2973.200	0.91	16.0/16.0	13.67	7.03
B50_1	2686.167	2720.000	1.22	**14/14**	28.60	20.79
B50_2	3514.500	3608.000	2.64	**17/17**	43.87	38.75
B50_3	2036.000	2069.000	1.62	15/13	6.94	0.92
B50_4	2675.667	2712.000	1.34	**14/14**	6.22	1.59
B50_5	2972.680	3014.000	1.37	**16/16**	6.10	2.24
B50_6	3107.333	3115.000	0.22	**15/15**	7.74	1.35
B50_7	3087.000	3088.000	0.03	**16/16**	9.46	1.88
B50_8	3790.000	3790.000	**0.00**	**18/18**	4.38	0.35
B50_9	2499.000	2499.000	**0.00**	**15/15**	10.38	0.16
B50_10	2969.833	2971.000	0.03	16/15	10.39	0.58
∅	2933.818	2958.600	0.84	15.6/15.3	13.40	6.86
DAR30_1	1357.250	1391.000	2.43	**8/8**	2.02	0.75
DAR30_2	1932.000	1932.000	**0.00**	**10/10**	0.70	0.09
DAR30_3	1342.000	1349.000	0.52	11/10	0.86	0.13
DAR30_4	1159.500	1249.000	7.67	**9/9**	2.89	1.75
DAR30_5	1667.400	1672.000	0.23	**8/8**	0.77	0.17
DAR30_6	2354.000	2418.000	2.71	**11/11**	31.85	31.21
DAR30_7	1355.500	1377.000	1.54	**8/8**	1.22	0.19
DAR30_8	1273.750	1291.000	1.33	10/9	0.71	0.17
DAR30_9	1856.250	1861.000	0.21	**9/9**	1.08	0.18
DAR30_10	1662.500	1667.000	0.24	**8/8**	1.50	0.13
∅	1596.015	1620.700	1.68	9.2/9.0	4.36	3.47
DAR50_1	1968.000	2069.000	5.13	15/12	49.42	44.59
DAR50_2	2555.389	2687.000	5.12	15/13	289.67	284.55
DAR50_3	1848.500	1907.000	3.13	12/14	25.04	15.74
DAR50_4	2190.091	2229.000	1.73	13/14	13.50	5.62
DAR50_5	1803.000	1900.000	5.37	**13/13**	46.68	36.95
DAR50_6	1943.929	2001.000	2.93	13/14	12.19	2.22
DAR50_7	2275.750	2304.000	1.23	14/13	9.72	2.70
DAR50_8	2056.000	2164.000	5.25	13/12	80.81	69.95
DAR50_9	3014.464	3141.000	4.17	14/12	21.81	14.38
DAR50_10	1799.833	1853.000	2.94	11/13	29.95	5.68
∅	2145.495	2225.500	3.70	13.3/13.0	57.87	48.23

Acknowledgments

This research was funded by the German Federal Ministry of Education and Research (BMBF) under Grant Number 03-ZI7BR2-1. The authors are grateful to EKO Trans GmbH, Verkehrsbetriebe Peine-Salzgitter GmbH, and Marc Sol for providing them with data sets. They thank Jacques Desrosiers for his careful reading of earlier drafts of this paper and for many fruitful discussions and guidance, which lead to a more focused presentation. The authors also appreciate a quick English lesson from Moshe Dror.

References

Ahuja, R. K., T. L. Magnanti, J. B. Orlin. 1993. *Network Flows: Theory, Algorithms and Applications*. Prentice-Hall, Englewood Cliffs, NJ.

Barnhart, C., E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, P. H. Vance. 1998. Branch-and-price: Column generation for solving huge integer programs. *Oper. Res.* **46** 316–329.

Charnes A., M. H. Miller. 1956. A model for the optimal programming of railway freight train movements. *Management Sci.* **3** 74–92.

Crainic T. G., J.-M. Rousseau. 1987. The column generation principle and the airline crew pairing problem. *INFOR* **25** 136–151.

Desaulniers, G., J. Desrosiers, M. M. Solomon. 2001. Accelerating strategies in column generation methods for vehicle routing and crew scheduling problems. C. C. Ribeiro, P. Hansen, eds. *Essays and Surveys in Metaheuristics*. Kluwer, Boston, MA, 309–324.

——, ——, I. Ioachim, M. M. Solomon, F. Soumis, D. Villeneuve. 1998. A unified framework for deterministic time constrained vehicle routing and crew scheduling problems. T. G. Crainic, G. Laporte, eds. *Fleet Management and Logistics*. Kluwer, Boston, MA, 57–93.

Desrochers M., F. Soumis. 1988a. A generalized permanent labelling algorithm for the shortest path problem with time windows. *INFOR* **26** 191–212.

——, ——. 1988b. A reoptimization algorithm for the shortest path problem with time windows. *European J. Oper. Res.* **35** 242–254.

Desrosiers J., Y. Dumas. 1988. The shortest path problem for the construction of vehicle routes with pick-up, delivery and time constraints. *Adv. Optimization and Control*, *Lecture Notes in Econom. Math. Systems* **302** 144–157.

——, ——, F. Soumis. 1986. A dynamic programming solution of the large-scale single-vehicle dial-a-ride problem with time windows. *Amer. J. Math. Management Sci.* **6** 301–326.

——, P. Pelletier, F. Soumis. 1983. Plus court chemin avec contraintes d'horaires. *RAIRO Recherche Opér.* **17** 357–377.

——, Y. Dumas, M. M. Solomon, F. Soumis. 1995. Time constrained routing and scheduling. M. O. Ball, T. L. Magnanti, C. L. Monma, G. L. Nemhauser, eds. *Network Routing, Handbooks in Operations Research and Management Science*, Vol. 8. North-Holland, Amsterdam, The Netherlands, 35–139.

Dumas, Y., J. Desrosiers, F. Soumis. 1991. The pickup and delivery problem with time windows. *Eur. J. Oper. Res.* **54** 7–22.

Gegen, C., P. Dejax, M. Dror, D. Feillet, M. Gendreau. 2000. An exact algorithm for the elementary shortest path problem with resource constraints. Technical Report CRT-2000-15, C.R.T. Montréal, Canada.

Ibaraki, T. 1987. *Enumerative Approaches to Combinatorial Optimization*, *Annals of Operations Research*, Vol. 10–11. Baltzer.

Lübbecke, M. E. 1999. Dual variable based fathoming in dynamic programs used for column generation. Technical report, Dept. Mathematical Optimization, Braunschweig University of Technology. Submitted.

——. 2001a. Combinatorial restrictions on pickup and delivery paths. Technical report, Dept. Mathematical Optimization, Braunschweig University of Technology. Submitted.

——. 2001b. Engine scheduling by column generation. Ph.D. thesis, Braunschweig University of Technology.

——, U. T. Zimmermann. 2003. Computer aided scheduling of switching engines. W. Jöger, H.-J. Krebs, eds. *Mathematics—Key Technology for the Future*: *Joint Projects Between Universities and Industry*. Springer, Berlin, Germany, 690–703.

Reingold, E. M., J. Nievergelt, N. Deo. 1977. *Combinatorial Algorithms—Theory and Practice*. Prentice-Hall, Englewood Cliffs, NJ.

Savelsbergh M. W. P., M. Sol. 1995. The general pickup and delivery problem. *Trans. Sci.* **29** 17–29.

——, ——. 1998. DRIVE: Dynamic routing of independent vehicles. *Oper. Res.* **46** 474–490.

Sol, M. 1994. Column generation techniques for pickup and delivery problems. Ph.D. thesis, Eindhoven University of Technology.

Received: January 2002; accepted: February 2002.