

REAL-TIME DISPATCHING OF GUIDED AND UNGUIDED AUTOMOBILE SERVICE UNITS WITH SOFT TIME WINDOWS

SVEN O. KRUMKE, JÖRG RAMBAU, AND LUIS M. TORRES

ABSTRACT. We investigate a real-world large scale vehicle dispatching problem with strict real-time requirements, posed by our cooperation partner, the German Automobile Association. We present computational experience on real-world data with a dynamic column generation method employing a portfolio of acceleration techniques. Our computer program ZIBDIP yields solutions on heavy-load real-world instances (215 service requests, 95 service units) in less than a minute that are no worse than 1% from optimum on state-of-the-art personal computers.

1. INTRODUCTION

The German Automobile Association *ADAC* (*Allgemeiner Deutscher Automobil-Club*), the second largest automobile club worldwide, maintains a heterogeneous fleet of over 1600 service vehicles in order to help people whose cars break down on their way. All service vehicles (*units*, for short) are equipped with GPS, which helps to exactly locate each unit in the fleet. In five ADAC help centers (*Pannenhilfezentralen*) spread over Germany, human operators (*dispatcher*) constantly assign units to incoming help requests (*events*, for short) so as to provide for a good quality of service (i.e., waiting times of less than 20–60 minutes depending on the system load) and low operational costs (i.e., short total tour length and little overtime costs). Moreover, about 5000 units of service contractors (*conts*, for short)—not guided by ADAC—can be employed to cover events that otherwise could not be served in time. This manual dispatching system is now subject to automation.

1.1. Informal Problem Description. Given a snapshot in the continuously running planning process, the task of the dispatcher in one of the help centers is to assign a unit or a contractor to each event and a tour to each unit such that every event is served by a unit or contractor that is capable of this service and such that a certain cost function is minimized. The result of this planning process is a (tentative) dispatch. The overall goal is to design an automatic online dispatching system that guarantees small waiting times for events and low operational costs when regarded over a larger period of time. In particular, the ADAC chose to impose a soft deadline on the service of an event that may be missed at the cost of a linearly increasing lateness penalty (*soft time windows*).

A basic building block for such a system is a fast *offline optimization module* that is able to produce an optimal or near-optimal dispatch under strict real-time requirements. More specifically, the optimization module must provide a reasonable answer in less than a second and must have the ability to improve on that solution whenever by some circumstances more time is granted to the optimization process. Given the fact that in an average snapshot 200 yet unserved events have to be assigned to tours for about 100 vehicles at distinct positions, the real-time aspect requires special attention.

1.2. Related Work. See [1] for evidence that the ability of fast reoptimization can help to improve the performance of dynamic dispatching systems. A possible method to organize the dynamics of a dispatching system in the language of *agents* can be found in [10]. Whether or not precomputed routes should be subject to sudden change are discussed in [4], where the optimization part is done by tabu search.

The basic problem can be modeled as a *multi depot vehicle routing problem with soft time windows* MVRPSTW, where each guided vehicle is a depot of its own and the contractors maintain a depot of a certain capacity. Many algorithms, heuristic and exact (i.e., where a performance guarantee can be given a-posteriori), have been proposed for the related *vehicle routing problem with time windows*, where the time windows have to be respected in any feasible dispatch (see [3] and references therein for a survey of various problem types and exact algorithms; see [5] for recent progress in efficiency of exact algorithms; see [9] for a tabu search approach dealing with soft time windows; see [7] for one approaches based on genetic algorithms).

To the best of our knowledge none of the exact algorithms was ever reported to *predictably* meet strict real time requirements in a large-scale real-world application, i.e., produce reasonable answers very early (after five seconds) in the course of the optimization process. On the other hand, the heuristic methods cannot guarantee a certain quality of the delivered solution.

1.3. Our Contribution. We will show in this paper that we can employ a custom-made dynamic column generation method (we call it *Dynamic Pricing Control*) to this problem that delivers good solutions after a fraction of a second and yields a provably optimal or near-optimal solution in less than five seconds in all real-world data sets with about 200 events and about 100 units provided by ADAC. The behaviour remains stable even for extremal load problems (artificially augmented real-world problems) with up to 770 events and 200 units: the solution quality was already within 12% from optimum after 5 seconds, within 5% from optimum after 15 seconds, and within 2% after one minute.

We had the chance to compare an implementation of our algorithm with an experimental prototype using meta-heuristics based on genetic algorithms and hill-climbing, which was produced by our industrial partner with serious effort; it turned out that four variants of the code based on meta-heuristics are clearly outperformed by our exact method on real-world data.

1.4. Outline. The rest of the paper is organized as follows: In the next section we introduce the exact setting of the VDP. Section 3 is devoted to the mathematical model that is the basis for the column generation approach. In Section 4 we describe our real-time compliant algorithm. Computational results on real-world data in Section 5 evaluate the effectiveness of various algorithmic tuning concepts. Section 6 summarizes the key points of this paper.

2. PROBLEM SPECIFICATION

In the following, we specify the exact form of VDP that is tackled by our algorithm. An instance of the VDP consists of a set of units, a set of contractors, and a set of events.

Each unit u has a current position o_u , a home position d_u , a logon time t_u^{start} , a shift end time t_u^{end} , and a set of capabilities F_u . Moreover, the costs related to using this unit are specified by values for costs per time unit for each of the following actions: driving c_u^{drv} , waiting c_u^{wait} , serving c_u^{svc} , and overtime c_u^{ot} .

Each contractor v has a home position d_v and a set of capabilities F_v . Moreover, the costs for booking the contractor are specified by a value for costs per service c_v^{svc} .

Each event e has a position x_e , a release time θ_e^r , a deadline θ_e^d , a service time δ_e , and a set of required capabilities F_e . Moreover, extra costs related to serving this event are specified by the value of a lateness coefficient c_e^{late} meaning that a cost of c_e^{late} times the delay w.r.t. the deadline of the event is incurred.

A feasible solution of the VDP (a *dispatch*) is an assignment of events to units and contractors capable of serving them, as well as a tour for each unit such that all events are assigned, the service of events does not start before their release times (waiting is allowed), and all tours for all units start at their current positions not before their logon times and end at their home positions. The costs of a dispatch are the sum of all unit costs, contractor costs, and event costs.

3. MODELING

We will use a model based on tour variables. Models of this type are by now well-established in the vehicle routing literature (see, e.g., [3]).

Let \mathcal{R} be the set of all feasible *tours*. This set splits into the sets \mathcal{R}_u of feasible tours for each unit u . A tour in \mathcal{R}_u can be described by an ordered sequence $(u, e_1, e_2, \dots, e_k)$ of k distinct events visited by u in that order. We will use the sequence (u) to denote the *go-home* tour, i.e., the tour in which u travels from its current position directly to its home position. Feasibility means that the capabilities of the unit are sufficient for e_i , i.e., $F_{e_i} \subseteq F_u$, for all $i = 1, \dots, k$. Notice that this sequence also fixes the arrival times of u at each event.

For all $R \in \mathcal{R}_u$ we introduce binary variables x_R with the following meaning: $x_R = 1$ if and only if the route R is chosen to be in the dispatch.

The cost of the route R is denoted by c_R and computed as follows. Let δ_u^{ef} be the driving time of unit u from event e to event f . Moreover, let $\delta_u^{o_u e}$ resp. $\delta_u^{e d_u}$ be the driving times of unit u from its current position to event e resp. from event e to its home position d_u . By t_R^e we denote the arrival time at event e in route R . The arrival time of u at its home position be $t_R^{d_u}$. Then the cost c_R of route $R = (u, e_1, e_2, \dots, e_k)$ can be computed as

$$\begin{aligned}
c_R &= c_u^{\text{drv}} \delta_u^{o_u e_1} + \sum_{i=2}^k c_u^{\text{drv}} \delta_u^{e_{i-1} e_i} + c_u^{\text{drv}} \delta_u^{e_k d_u} && \text{(driving)} \\
&+ \sum_{i=1}^k c_u^{\text{svc}} \delta_{e_i} && \text{(service)} \\
&+ c_u^{\text{ot}} \max\{(t_R^{d_u} - t_u^{\text{end}}), 0\} && \text{(overtime)} \\
&+ \sum_{i=1}^k c_{e_i}^{\text{late}} \max\{(t_R^{e_i} - \theta_{e_i}^d), 0\} && \text{(lateness)}
\end{aligned}$$

A feasible “route” S for a contractor v can be written as a set $\{e_1, e_2, \dots, e_k\}$ of events that this contractor may be assigned to serve, i.e., $F_{e_i} \subseteq F_v$ for all $i = 1, \dots, k$.

Let t_v^e be the time by which contractor v can have reached event e with one of his vehicles. The cost c_S of such a “tour” S can be computed as follows:

$$c_S = c_v^{\text{svc}} |S| \quad (\text{service})$$

$$+ \sum_{i=1}^k c_{e_i}^{\text{late}} \max\{(t_v^{e_i} - \theta_{e_i}^d), 0\} \quad (\text{lateness})$$

Since this cost is linear in the events served by this contractor, every “tour” S of a contractor can be combined from elementary contractor tours containing each only a single event.¹ Let \mathcal{S}^v be the set of elementary feasible “tours” for v , and let \mathcal{S} be their union over all contractors $v \in V$.

For all $S \in \mathcal{S}^v$ we introduce binary variables x_S with the following meaning: $x_S = 1$ if and only if the elementary “tour” S is chosen to be in the dispatch.

The VDP can now be formulated as a set partitioning problem as follows. Let a_{Re}, b_{Se} be binary coefficients with $a_{Re} = 1$ (resp. $b_{Se} = 1$) if and only if event e is served in tour R (resp. in elementary contractor “tour” S).

$$\min \sum_{R \in \mathcal{R}} c_R x_R + \sum_{S \in \mathcal{S}} c_S x_S \quad (\text{IP})$$

subject to

$$\sum_{S \in \mathcal{S}} b_{Se} x_S + \sum_{R \in \mathcal{R}} a_{Re} x_R = 1 \quad \forall e \in E; \quad (1)$$

$$\sum_{R \in \mathcal{R}_u} x_R = 1 \quad \forall u \in U; \quad (2)$$

$$x_R \in \{0, 1\} \quad \forall R \in \mathcal{R}; \quad (3)$$

$$x_S \in \{0, 1\} \quad \forall S \in \mathcal{S}. \quad (4)$$

Our method is based on solving the linear programming relaxation (LP) of (IP) by dynamic column generation.

We would like to estimate during the column generation process how far we are still away from the optimal solution of (LP). To this end, we use a bound [6] coming from the Langrangean relaxation of (LP) w.r.t. the constraints (1).

Lemma 3.1. *Let (π_e^*, π_u^*) be an optimal solution to the dual of the reduced LP, and let $(x_R^*, x_S^*)^T$ be the corresponding primal solution. Then the cost $c_{\text{LP}}^{\text{opt}}$ of an optimal solution of the LP satisfies*

$$c_{\text{LP}}^{\text{opt}} \geq \sum_{R \in \tilde{\mathcal{R}}} c_R x_R^* + \sum_{S \in \mathcal{S}} c_S x_S^* + \sum_{u \in U} \min_{R \in \mathcal{R}_u} (c_R - \sum_{e \in E} a_{Re} \pi_e^* - \pi_u^*) \quad (5)$$

□

¹So far, there are no data about the capacities of contractors available to the ADAC. Thus, an infinite capacity is assumed. Later, in the dynamic optimization process, a contractor that declines a request will be removed from the dispatching system for some time.

This lower bound is useful in the course of a column generation algorithm since its main terms have to be computed during the column generation process anyway.

4. THE ALGORITHM

The input of the top level algorithm in ZIBDIP is an instance of the VDP. The initial LP consists of all elementary tours for all contractors plus a tour for each unit from its current position to its home position (*go-home-tour*). This way, both the initial LP and the initial IP are feasible.

The search for additional columns is done in a *depth-first-search branch&bound tree* (*search tree*, for short) for each unit. Each node in the search tree corresponds to a tour starting at the current position of a unit and ending at the position of the last event served by the unit. A node can be completed to a feasible tour by appending the tour from the position of the last event in the node to the unit's home position. The *pre-cost* of a node in the search tree is the *reduced cost* [2, 8] of the corresponding tour (without returning to the unit's home position and overtime). The *cost* of a node in the search tree is defined as the reduced cost of the corresponding feasible tour (including the costs for returning to the home position and overtime). The *dual prices* for the events and units are taken from the previous run of the LP solver.

The root node r of the search tree corresponds to the empty tour. Given a node v in the tree, the children of v are obtained by appending one event to v that is not yet in v .

We generate columns for each unit in loops with increasing values for the maximal search depth (inner loop) and the maximal search degree (outer loop). The values for the maximal search depth are increased until no progress has been made in the previous step provided the search depth was sufficiently large, at latest when the depth equals the number of events. The search degree is increased until an optimality criterion is met or the search degree has reached the number of events.

While we are adding columns to the LP we fix the upper bound to a negative acceptance threshold: all columns that have reduced costs smaller than the acceptance threshold are added to the LP. This acceptance threshold is updated after each iteration depending on the number of columns produced.

This search on a dynamically growing space ensures that

- the effort of finding new columns is small in the beginning, when the dual variables are not yet in good shape
- the dual variables are updated often in the beginning
- this update is fast since the number of columns in the LP is still small
- we can enforce the output of a feasible integer solution early
- the search is exact later in the run when the dual information is reliable

Whenever a new integral solution is found we output the corresponding dispatch.

5. COMPUTATIONAL RESULTS ON REAL-WORLD DATA: EFFECT OF DYNAMIC PRICING CONTROL

In order to show the effectiveness of the ZIBDIP algorithm, we compared the following setups on the real-world data sets of Table 1:

- an *unmodified* column generation procedure, where the pricing step is done on the whole search space and all columns with negative reduced costs are included in the new reduced LP (*all-off*)

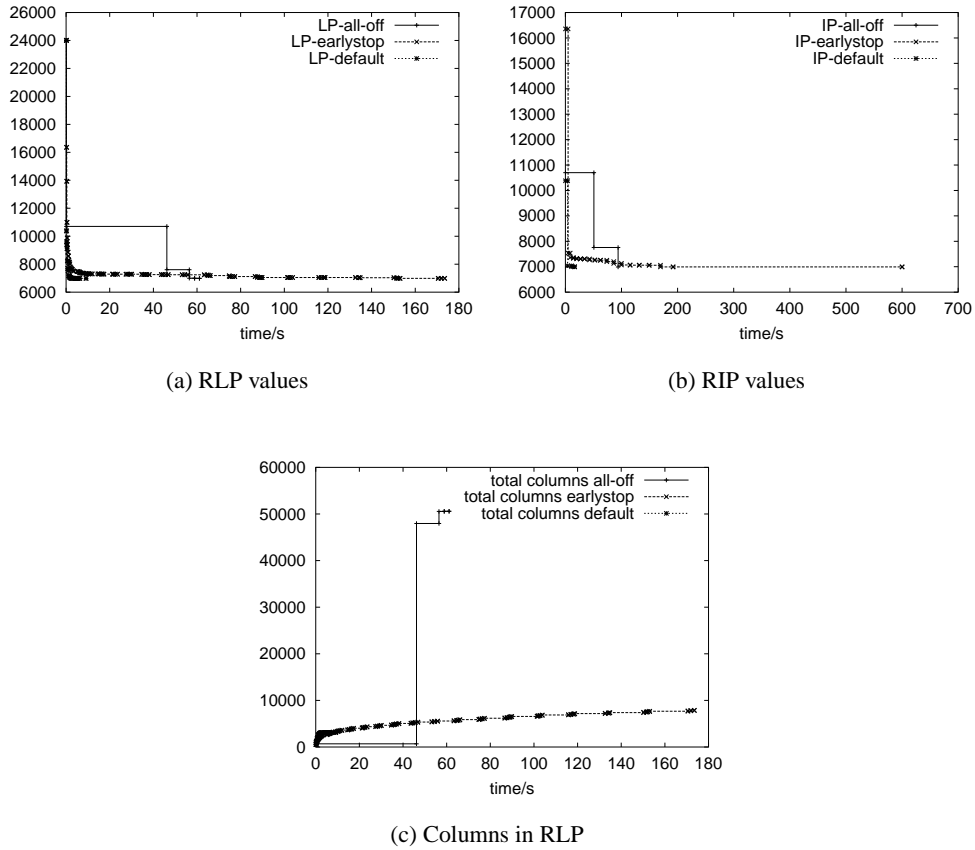


FIGURE 1. Results for 2702_high

- *forced early stop* in the pricing step: whenever one column with negative reduced costs is found then the pricing step is interrupted, and the new column is added to the reduced LP (this method was successfully applied by [5] on modified instances of some Solomon problems (*earlystop*)),
- the default settings of ZIBDIP (*default*).

Data File	System Load	# events	# units
2702_high	high	215	98
Xtreme	simulated extreme load	775	211
prob700	randomized extreme load	700	100

TABLE 1. Overview over the data sets used in the evaluation.

We have plotted over time the development of the optimal solutions of the reduced LPs (RLP) and the corresponding IPs (RIP) as well as the accumulated number of columns.

The results in Figures 1 and 2 show that a vanilla column generation algorithm is not capable to meet the real time requirements: far too many generated columns lead

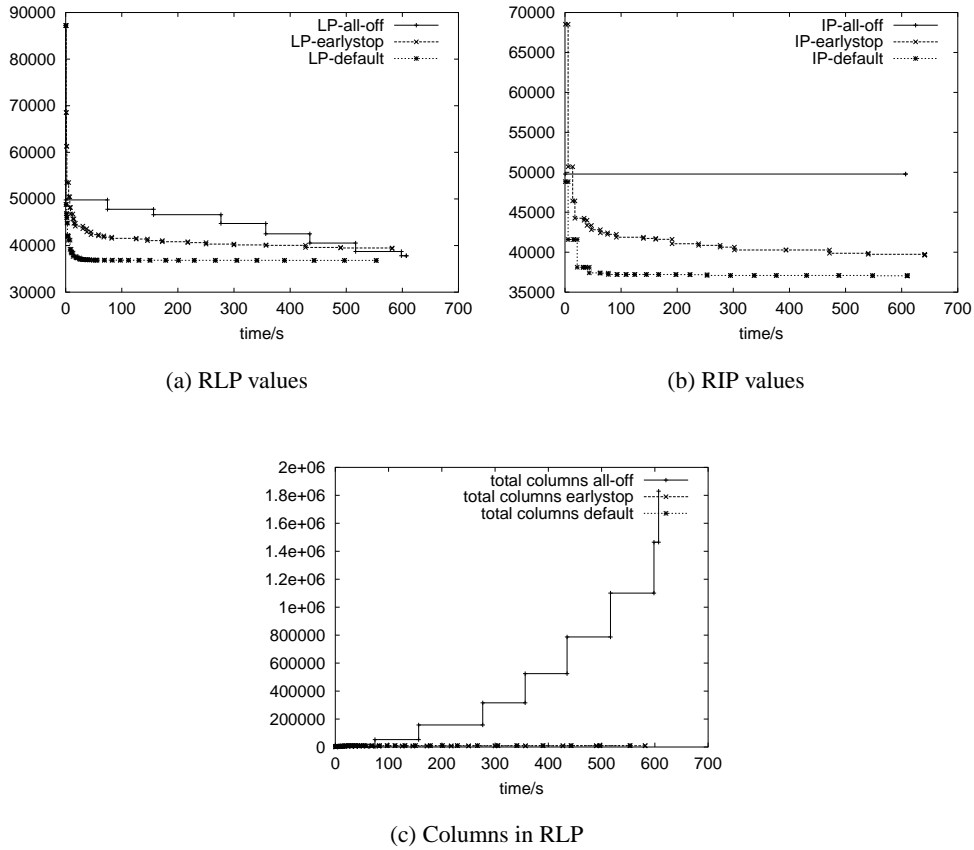


FIGURE 2. Results for Xtreme

to an unacceptable solution time both in pricing and in LP solving. This, by the way, does not change if the column generation step is interrupted after the optimal column was found. It can be seen that *forced early stop* is for our special kind of problem not the proper approach to deal with this difficulty: too many iterations are wasted by adding columns with inferior quality, and so *forced early stop* is clearly outperformed by ZIBDIP's default settings.

Observe in Figure 1 that a successful optimality check for the default setting of ZIBDIP made the program terminate already within 15 seconds whereas the all-off and the earlystop settings need a great deal longer to nail down the optimal LP solution.

We have run this test on all of our data with the same results, except that for the low load instances the differences were not equally substantial.

For a more detailed look at the effects of specific algorithmic features of ZIBDIP, we compared the performances of

- *forced early stop* (earlystop),
- ZIBDIP in default settings (default),
- ZIBDIP with a node-2-OPT start heuristics (default_heu2),

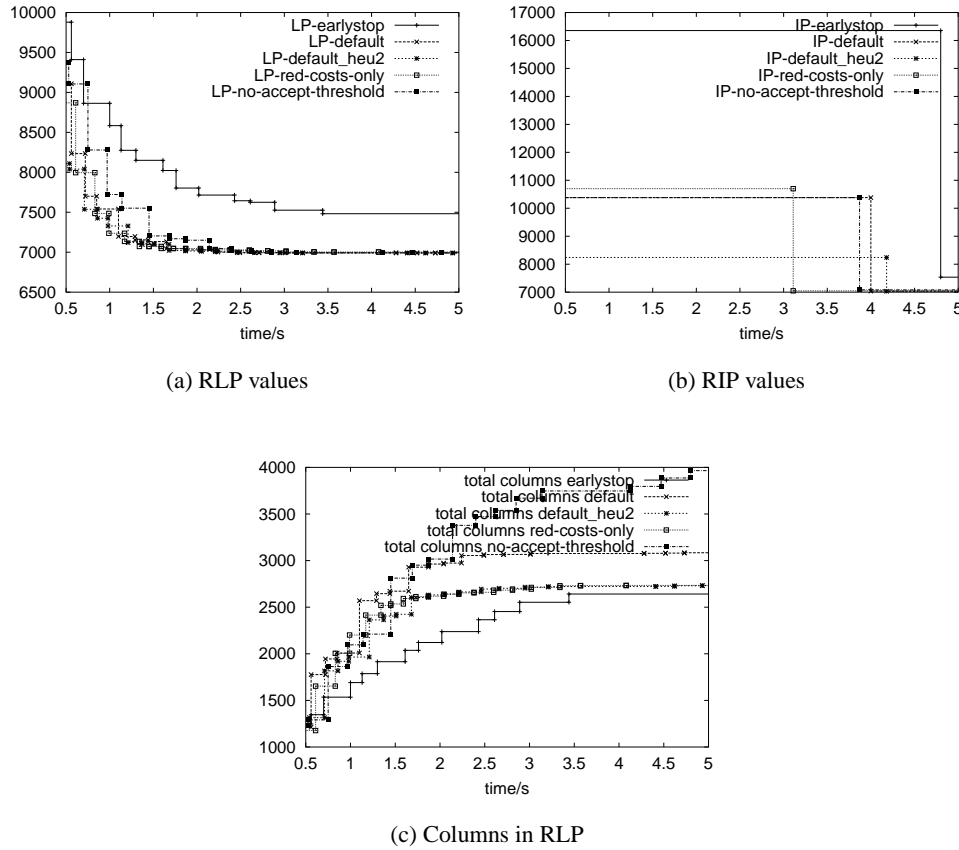


FIGURE 3. Results for 2702_high

- ZIBDIP the acceptance threshold is set to zero throughout, thus all tours with negative reduced costs found in the restricted searchspace are accepted (no-accept-threshold),
- ZIBDIP without the variation of sorting criteria in the pricing step: tours are always extended in the order of increasing reduced costs rather than according to alternating sorting criteria (red-costs-only).

We concentrate on the early phase that is most important for the realtime-application (5 seconds for high load instances, 15 seconds for extreme load instances). Figures 3 through 5 show the performance indicated by the objective values and the number of generated columns.

We add one additional set of input data `prob700` with 700 events and 100 units. It was created randomly and has release times in the future. It serves as a stability checker for our methods since we expect that its uniform structure and relatively small late penalties allow for longer tours and make it harder for ZIBDIP to find the optimal tours.

The results in Figures 3 through 5 show that using the starting heuristics usually improves the IP solutions of ZIBDIP during the first five seconds. The convergence of

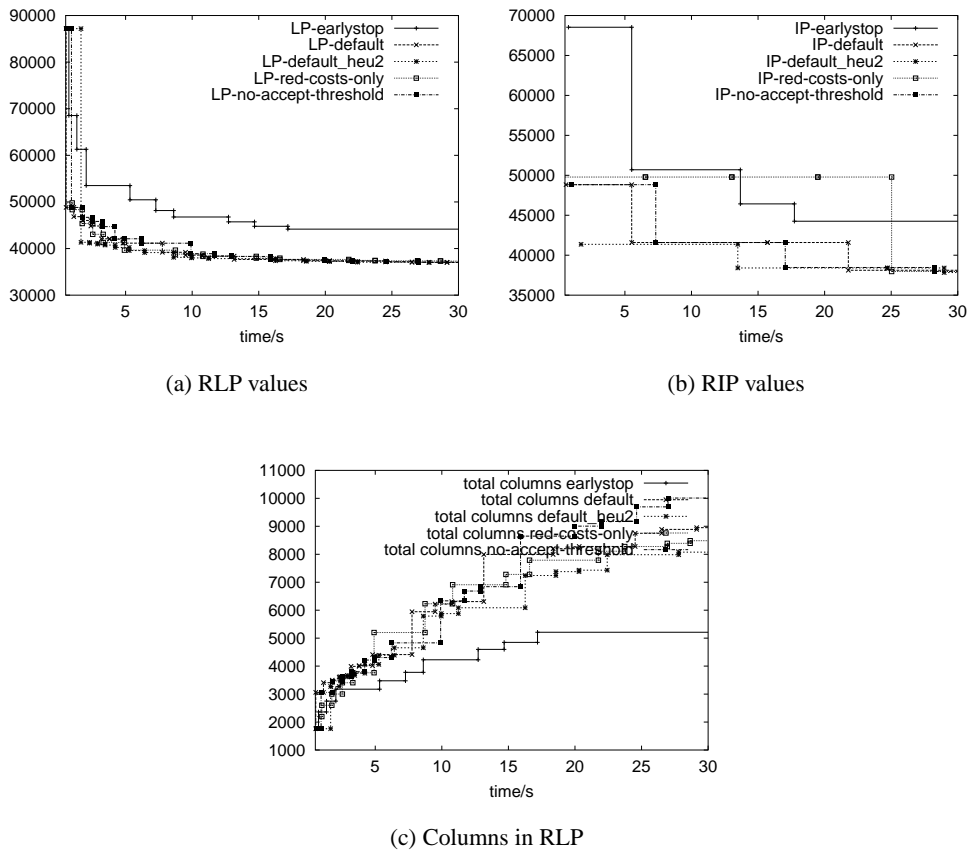


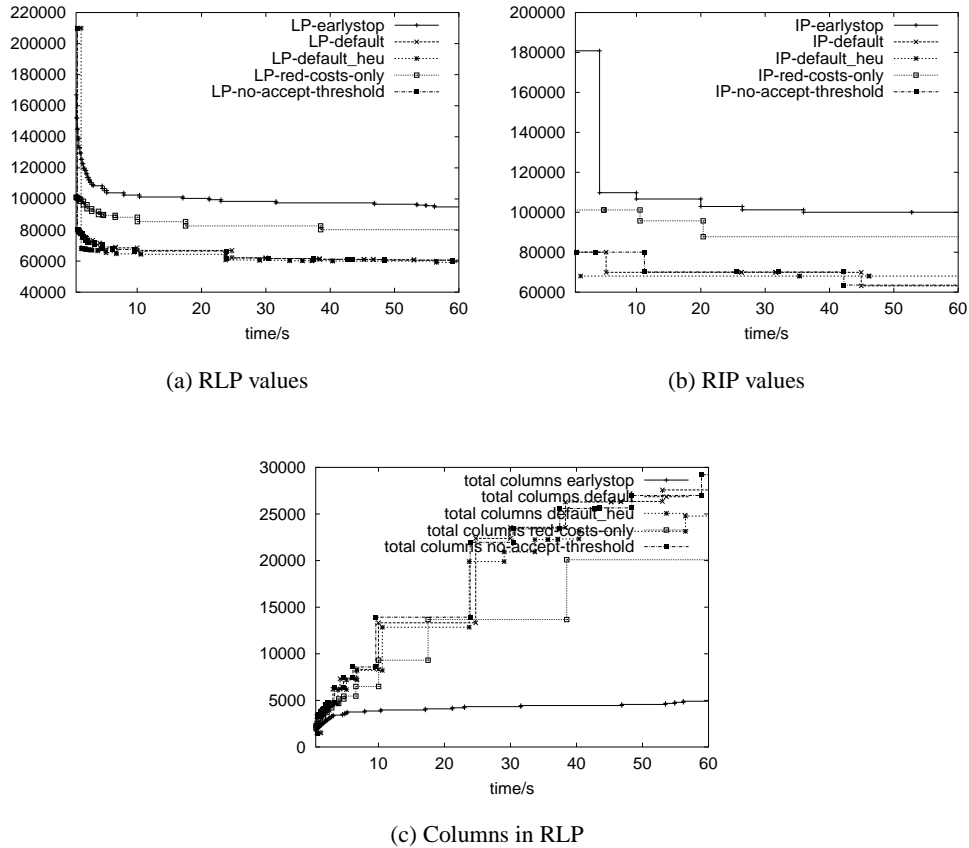
FIGURE 4. Results for Xtreme

the further column generation procedure is, however, not affected by the start heuristics.

Dropping the dynamic acceptance threshold leads to more columns resulting in a slightly worse performance in the speed of LP convergence. This behaviour is, however, by far not as significant as the restriction of the search space: once the search space is restricted acceptance control of new columns is a fine tuning issue.

The influence of the variation of sorting criteria, while neglectable on the real-world data, is strong in the random data example. This is plausible because a greedy search by reduced costs is more promising in the absence of long tours. Since `prob700` has tours of length 9 in its near-optimal solution while for `Xtreme` there are only tours of length 4, a pure greedy search is to “narrow-minded” to find the best tours early.

Anyway: since varying the sorting criterion does not harm in the other test cases it seems advisable to use it in order to be weaponed against pathologic input data.

FIGURE 5. Results for `prob700`

6. CONCLUSION

We have presented the specialized column generation algorithm **ZIBDIP** that solves a real-world large scale vehicle dispatching problem with soft time windows under real-time requirements. The problem arises as a subproblem in an online-dispatching task that was proposed to us by the German Automobile Association (ADAC). The algorithm clearly outperforms an experimental prototype code based on primal heuristics and genetic algorithms provided by our industrial partner. Moreover, **ZIBDIP** is able to provide a lower bound based on an optimal LP solution in seconds for all real-world instances provided by ADAC. It was shown that the concept of Dynamic Pricing Control can significantly speed up convergence of the column generation process, thereby making a method that has proven to be effective for large scale offline problems ready for the use in online-algorithms under realtime requirements. The practical impact of this work is that **ZIBDIP** is being reimplemented into the new commercial standard automatic dispatch system distributed by IPS, one of the main providers of dispatching software, superseding the former code based on primal meta-heuristics. Moreover: this product will finally be used in ADAC's help centers.

REFERENCES

1. Julien Bramel and David Simchi-Levi, *On the effectiveness of set covering formulations for the vehicle routing problem with time windows*, *Operations Research* **45** (1997), no. 2, 295–301.
2. Vasek Chvatal, *Linear programming*, Freeman, New York, 1983.
3. Jaques Desrosiers, Yvan Dumas, Marius M. Solomon, and François Soumis, *Time constraint routing and scheduling*, *Network Routing* (Michael Ball, Tom Magnanti, Clyde Monma, and George Newhauser, eds.), *Handbooks in Operations Research and Management Science*, vol. 8, Elsevier, Amsterdam, 1995, pp. 35–140.
4. Soumia Ichoua, Michel Gendreau, and Jean-Yves Potvin, *Diversion issues in real-time vehicle dispatching*, *Transportation Science* **34** (2000), no. 4, 426–438.
5. Jesper Larsen, *Vehicle routing with time windows—finding optimal solutions efficiently*, *DORSnyt* (engl.) (1999), no. 116.
6. Leon S. Lasdon, *Optimization theory for large systems*, Macmillan, New York, 1970.
7. Sushil J. Louis, Xiangying Yin, and Zhen Ya Yuan, *Multiple vehicle routing with time windows using genetic algorithms*, 1999 Congress on Evolutionary Computation (Piscataway, NJ), IEEE Service Center, 1999, pp. 1804–1808.
8. D. G. Luenberger, *Linear and nonlinear programming*, 2 ed., Addison-Wesley, 1984.
9. Éric Taillard, P. Badeau, Michel Gendreau, F. Guertin, and Jean-Yves Potvin, *A tabu search heuristic for the vehicle routing problem with soft time windows*, *Transportation Science* **31** (1997), 170–186.
10. Kenny Qili Zhu and Kar-Loon Ong, *A reactive method for real time dynamic vehicle routing problem*, *Proceedings of the 12th ICTAI*, 2000.

KONRAD-ZUSE-ZENTRUM FÜR INFORMATIONSTECHNIK BERLIN, TAKUSTR. 7, 14195 BERLIN,
GERMANY