Konrad-Zuse-Zentrum
für Informationstechnik Berlin

Takustraße 7
D-14195 Berlin-Dahlem
Germany

BENJAMIN HILLER SVEN O. KRUMKE JÖRG RAMBAU

Reoptimization Gaps versus Model Errors in Online-Dispatching of Service Units for ADAC

ZIB-Report 04-17 (April 2004, revised December 2004)

REOPTIMIZATION GAPS VERSUS MODEL ERRORS IN ONLINE-DISPATCHING OF SERVICE UNITS FOR ADAC

BENJAMIN HILLER, SVEN O. KRUMKE, AND JÖRG RAMBAU

Abstract. Under high load, the automated dispatching of service vehicles for the German Automobile Association (ADAC) must reoptimize a dispatch for 100–150 vehicles and 400 requests in about ten seconds to near optimality. In the presence of service contractors, this can be achieved by the column generation algorithm ZIBDIP. In metropolitan areas, however, service contractors cannot be dispatched automatically because they may decline. The problem: a model without contractors yields larger optimality gaps within ten seconds. One way out are simplified reoptimization models. These compute a short-term dispatch containing only some of the requests: unknown future requests will influence future service anyway. The simpler the models the better the gaps, but also the larger the model error. What is more significant: reoptimization gap or reoptimization model error? We answer this question in simulations on real-world ADAC data: only the new models ShadowPrice and $\text{ZIBDIP}_{\text{dummy}}$ can keep up with ZIBDIP.

1. Issues and Motivation

Currently, the German Automobile Association (ADAC) evaluates an automated dispatching system for service vehicles (units) and service contractors (contractors) on the basis of exact cost-reoptimization. This means that a current dispatch is maintained, which contains all known yet unserved requests and which is near optimal on the basis of the current data; whenever a unit becomes idle its next request is read from the current dispatch; at each event (new request, finished service, etc.) the dispatch is updated by a reoptimization run.

A feasible current dispatch for all known requests and available service vehicles is a partition of the requests into tours for units and contractors such that each request is in exactly one tour and each unit drives exactly one tour (maybe directly to its home position) so that the cost function is minimized. Cost contributions come from driving costs for units, fixed service costs per requests for contractors, and a strictly convex lateness costs for violation of soft time windows at each request (currently quadratic). The latter cost structure is chosen so as to avoid large individual waiting times for customers.

It is not a-priori clear that such a rigorous reoptimization yields the best, or even a good, long-term cost (the *online issue* of the dispatching problem). Indeed, at times in the literature it is claimed that exact reoptimization (i.e., with

Key words and phrases. vehicle dispatching, real-time, integer linear programming, dynamic column generation, dummy contractor, shadow price model.

The first author was supported by the DFG Research Group "Algorithms, Structure, Randomness".

Work supported by the DFG Research Center Matheon "Mathematics for key technologies" in Berlin.

small optimality gap) does not pay in practice because of the unknown future requests [1, p. 5]. In the case of this particular application, however, the results of exact reoptimization are satisfying [2], in concordance with [3, Sec. 8.4].

Although the reoptimization problem, which is modeled as a set partitioning problem for tours, has an astronomical number of variables, it can be solved by a dynamic column generation procedure. An effective method to obtain provably good solutions in ten seconds (the *real-time aspect* of the dispatching problem) is *dynamic pricing control*, which is the main feature of our ZIBDIP algorithm (a thourough description of the algorithm and computational results can be found in [4]).

As it turns out, the fixed costs for service by contractors bound the dual values of requests. Thus, contractors substantially contribute to the success of ZIBDIP. The contractor, however, may in practice decline to serve suggested requests, in which case this request has to be manually reentered into the system: a time consuming process. In metropolitan areas, contractors decline so often that the ADAC decided to remove contractors from the model.

In simulations on ADAC production data (three days in December 2002 with high load) without contractors, we encountered a significant reoptimization gap. For 2002/12/13, e.g., Fig. 1 shows the gap of the reoptimization result to the respective lower bound coming from the optimal solution of the LP relaxation (this lower bound was computed a-posteriori for each reoptimization). The reoptimization still works well in most cases, but under high load the solutions – delivered after ten seconds – exhibit optimality gaps around 3% on average but up to 10% in peak load situations.

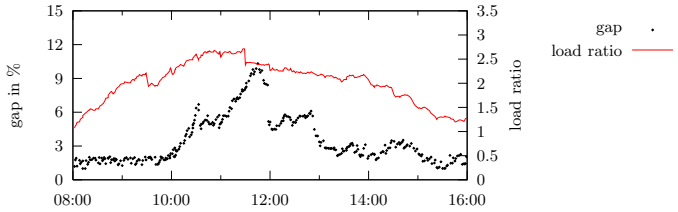


FIGURE 1. Optimality gap over time of ZIBDIP (the load ratio is the number of requests per unit in a reoptimization problem)

One way to overcome this problem is to consider *simplified reoptimization models* that stem from the following considerations: In principle, for each unit we only have to determine the next request to work on. The complete dispatch is computed only to pick up future synergies by considering more than one request per unit. Synergies that are implemented only very far in the future will be disturbed by new requests anyway; therefore, an exact pre-calculation of the best decisions in, say, two hours may not really be necessary; consequently, one can try to cover only *a subset of requests* in a reoptimization step.

The issue of this experimental work is: should one stick to the complete model and accept occasional substantial reoptimization gaps, or is it better to simplify the reoptimization model so as to eliminate the reoptimization gap? This question is answered on the basis of simulation studies, performed on the aforementioned ADAC production data: we first compare the original ZIBDIP reoptimization to several methods to select subsets of requests that have to be covered by any solution

of the reoptimization run. Then ZIBDIP competes with two simple online heuristics for the ZIBDIP model in order to estimate how even larger reoptimization gaps harm in the long run.

2. Simplified Models

We developed and evaluated the following strategies for the selection of requests to be covered in a reoptimization run. In the sequel, we describe the original and each simplified model in more detail.

In all these models, there is a binary selection variable x_T for each feasible tour T. Such a tour is given by a unit u and a sequence of requests to be served by u in the given order. We call the set of all feasible tours $\mathcal{T}$ and the set of all feasible tours for unit u is written as $\mathcal{T}_u$.

We denote by c_T the cost coefficient of tour T. This is a weighted sum of strictly convex late costs, linear drive costs, and strictly convex overtime costs. Late costs in the reoptimization are incurred whenever a request is served after a waiting time of more than 15 min. The true target for the waiting time is higher. The 15 min deadline in the reoptimization problem was derived from the following consideration: the true waiting time for a request should lead to the same lateness costs as the fixed contractor costs for serving that request. This is motivated by the wish that requests that can not be served inside the true time window by a unit should be served by a contractor in order to reach the true target time. The exact formula including the numerical values of the coefficients of the cost function can not be disclosed here.

Let (a_{vT}) be the incidence matrix of requests and tours.

2.1. The Original Model ZIBDIP.

The original reoptimization problem solved by ZIBDIP without contractors reads as follows.

$$\min \sum_{T \in \mathcal{T}} c_T x_T \quad \text{s.t.}$$

$$\text{(Partitioning Requests)} \qquad \sum_{T \in \mathcal{T}} a_{vT} x_T = 1 \qquad \forall \text{requests } v$$

$$\text{(Partitioning Units)} \qquad \sum_{T \in \mathcal{T}_u} x_T = 1 \qquad \forall \text{units } u$$

$$\text{(Binary Variables)} \qquad x_T \in \{0,1\} \qquad \forall T \in \mathcal{T}$$

This model guarantees that, after every reoptimization, each request is assigned to exactly one unit because of the set partitioning constraint. Every unit has to drive exactly one tour, where the direct move to its home position is also a feasible tour, the *drive-home tour*.

2.2. The Simplified Model 4-ZIBDIP.

Select those requests that are among the four closest to some unit. This can be generalized to k-ZIBDIP. In the following, *k-close requests* are requests that are among the k closest to some unit. In formulae,

we obtain the following model:

$$\min \sum_{T \in \mathcal{T}} c_T x_T \quad \text{s.t.}$$

(Partitioning k-Close Requests)	$\sum_{T \in \mathcal{T}} a_{vT} x_T = 1$	$\forall k$-close requests v
(Partitioning Units)	$\sum_{T \in \mathcal{T}_u} x_T = 1$	$\forall$units u
(Binary Variables)	$x_T \in \{0,1\}$	$\forall T \in \mathcal{T}$

Since this model does not guarantee the service for all requests even under low load, one has to switch back and forth between ZIBDIP and 4-ZIBDIP. In our experiments, we switch to 4-ZIBDIP whenever the load ratio surpassed a value of 2; we switch back to ZIBDIP whenever the load ratio drops below a value of 2.

2.3. **The Simplified Model PTC (Prescribed Total Cover).** Relax the set partitioning condition to set packing, and require that a request set of cardinality twice the number of units is covered by tours of units. This leads to the following model, where n_T is the number of requests in tour T:

$$\min \sum_{T \in \mathcal{T}} c_T x_T \quad \text{s.t.}$$

(Packing Requests)	$\sum_{T \in \mathcal{T}} a_{vT} x_T \leq 1$	$\forall$requests v
(Cardinality)	$\sum_{T \in \mathcal{T}} n_T x_T \geq 2\lvert\text{units}\rvert$	
(Partitioning Vehicles)	$\sum_{T \in \mathcal{T}_u} x_T = 1$	$\forall u \in U$
(Binary Variables)	$x_T \in \{0,1\}$	$\forall T \in \mathcal{T}$

This model requires at least twice as many requests as units in the system, which is the case under high load; under low load this may be infeasible, so one has to switch back and forth between ZIBDIP and PTC accordingly. In our experiments, we switch to PTC whenever the load ratio surpassed a value of 2; we switch back to ZIBDIP whenever the load ratio drops below a value of 2.

2.4. **The Simplified Model ShadowPrice.** Solve the LP relaxation of ZIBDIP. To find an integral solution, relax the set partitioning condition to set packing and change the cost of each tour to its reduced cost from the hopefully near optimal LP solution. In the following, the new cost coefficient $\tilde{c}^T$ of a tour T is the *reduced cost* of Tour T w. r. t. the best LP solution that can be found in time. Because the LP solution algorithms works by dynamic column generation, this solution is an optimal solution to the last RLP that could be solved in time. The resulting model

reads as follows:

$$\min \sum_{T \in \mathcal{T}} \tilde{c}_T x_T \quad \text{s.t.}$$

$$\text{(Packing Requests)} \qquad \sum_{T \in \mathcal{T}} a_{vT} x_T \leq 1 \qquad \forall \text{requests } v$$

$$\text{(Partitioning Units)} \qquad \sum_{T \in \mathcal{T}_u} x_T = 1 \qquad \forall \text{units } u$$

$$\text{(Binary Variables)} \qquad x_T \in \{0,1\} \qquad \forall T \in \mathcal{T}$$

In this model, requests are assigned to units only if their LP dual prices together with the drive-home cost of a unit pay enough to weigh out the primal costs of their service. This requires that the LP relaxation can be solved fast, since the LP is not simplified at all.

2.5. **The Simplified Model ZIBDIP$_{\text{dummy}}$.** Introduce a dummy contractor. This contractor can be assigned arbitrarily many requests at the same time at no extra cost, i.e., in reality, these requests are unassigned for the moment. In order to enforce a cost for the assignment to the dummy contractor, its arrival time at any request is a fixed time, the *dummy contractor delay*. In our case, 135 min were chosen. In the following, d_v is the dummy contractor delay, i.e., the late cost for 135 min additional delay at v (on top of the current age of v). This leads to the following model:

$$\min \sum_{T \in \mathcal{T}} c_T x_T + \sum_{v \in \text{requests}} d_v x_v \quad \text{s.t.}$$

$$\text{(Partitioning Requests)} \qquad \sum_{T \in \mathcal{T}} a_{vT} x_T + \sum_{v \in \text{requests}} x_v = 1 \qquad \forall \text{requests } v$$

$$\text{(Partitioning Units)} \qquad \sum_{T \in \mathcal{T}_u} x_T = 1 \qquad \forall \text{units } u$$

$$\text{(Binary Variables)} \qquad x_T \in \{0,1\} \qquad \forall T \in \mathcal{T}$$

This model implies that, in an optimal solution, for any request in a tour of a unit, service will start after at most 135 minutes after reoptimization; otherwise, the request would have been assigned to the dummy contractor.

3. Simplified Reoptimization Algorithms

We furthermore evaluated two heuristics for the original model, which where used in the reoptimization process as replacements for ZIBDIP. One should mention that in each reoptimization with either model, the solutions of the previous reoptimization are reused as start solutions – a simple but essential technique to stabilize the dispatching process in case of occasional suboptimal reoptimization.

3.1. **The Simplified Algorithm BestInsert.** A new dispatch is obtained by taking the dispatch of the previous reoptimization, removing all requests that have been served in the meantime, and inserting new requests at minimal additional cost w. r. t. to the original ZIBDIP-model.

3.2. **The Simplified Algorithm 2-Exchange.** A first tentative dispatch is computed by BestInsert. This dispatch is then improved by successively exchanging two requests between distinct time slots in the dispatch if this decreases the cost. It has to be noted that the complicated cost function for tours leads to quite some computational effort for the calculation of the 2-Exchange solutions.

4. Computational Results

The simulation data stems from three days of production at ADAC in December 2002; instance sizes are given in Table 1. Depending on the instance, between 1700 and 2100 reoptimization runs where triggered.

instance	*requests*	*units*	*requests per unit*
2002/12/07	2123	125	16.98
2002/12/13	2537	146	17.38
2002/12/14	1731	131	13.21

Table 1. Sizes of high load instances used for simulation

The software ran on a standard Linux PC, 2.4GHz Pentium 4 CPU, 4GB RAM, distribution Suse 9.0, kernel 2.4.21-202-smp, LP solver CPLEX 8.0, compiled with gcc 3.3.1. Each reoptimization run was interrupted after 10 seconds run-time.

4.1. **Simplified Models.** First of all, we checked whether the simplified models can reduce the optimality gaps of the reoptimization solutions that could be computed in 10 s (see Fig. 2). It can be seen that all models reduce the gap significantly, i.e., the corresponding optimization problems are easier to solve in 10 s.

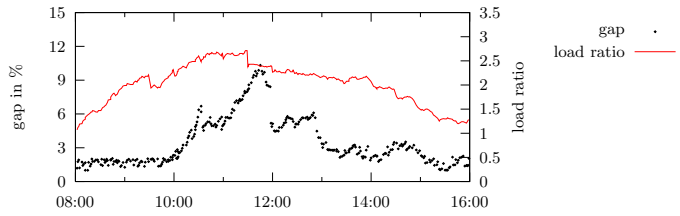
It has to be noted, though, that for 4-ZIBDIP and PTC the simplified model was only in effect for load ratios above two. Some single large optimality gaps stem from switches back to ZIBDIP because ZIBDIP has to run essentially without a start solution. This discontinuity in operation is certainly a draw-back of 4-ZIBDIP and PTC.

Next, we investigated the cost over time w. r. t. the reoptimization cost function, designed in cooperation with ADAC.

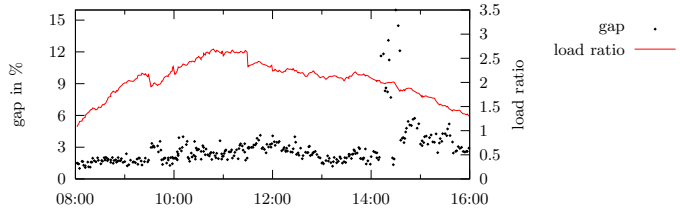
The results: only ShadowPrice and $\text{ZIBDIP}_{\text{dummy}}$ are competitive against ZIBDIP, although ShadowPrice seems to degrade in performance in the largest instance (b). In two out of three instances, ShadowPrice and $\text{ZIBDIP}_{\text{dummy}}$ have even slightly lower long-term cost than ZIBDIP, though by a small margin. In the largest instance with the most difficult reoptimization problems, however, the original ZIBDIP is superior. On average, however, the results are in favor of $\text{ZIBDIP}_{\text{dummy}}$.

Since the reoptimization cost function of ADAC is quite a complicated mixture of late, drive, and overtime costs, we decided to investigate two standard measures on the so-called late time vectors (see Fig. 4 and 5). The late time of a request is its waiting time portion that exceeds the allowed waiting time, fixed by ADAC. We calculated the L_1 norms and the L_2 norms of the late time vectors (one entry for each request). The former norms measure the avarage waiting time, the latter norms penalize in particular large individual late times, which is desirable from a fairness point of view.

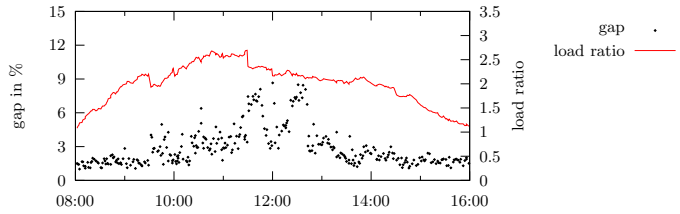
It is apparent, that w. r. t. these late time measures, $\text{ZIBDIP}_{\text{dummy}}$ is never worse than second best; moreover, it performs best in four out of six evaluations.



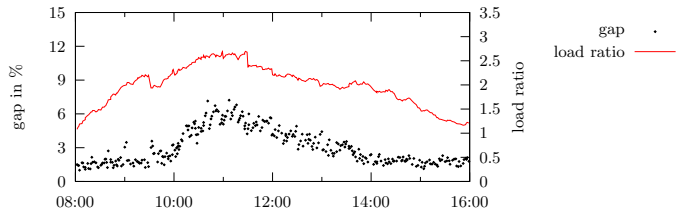
(a) ZIBDIP



(b) 4-ZIBDIP



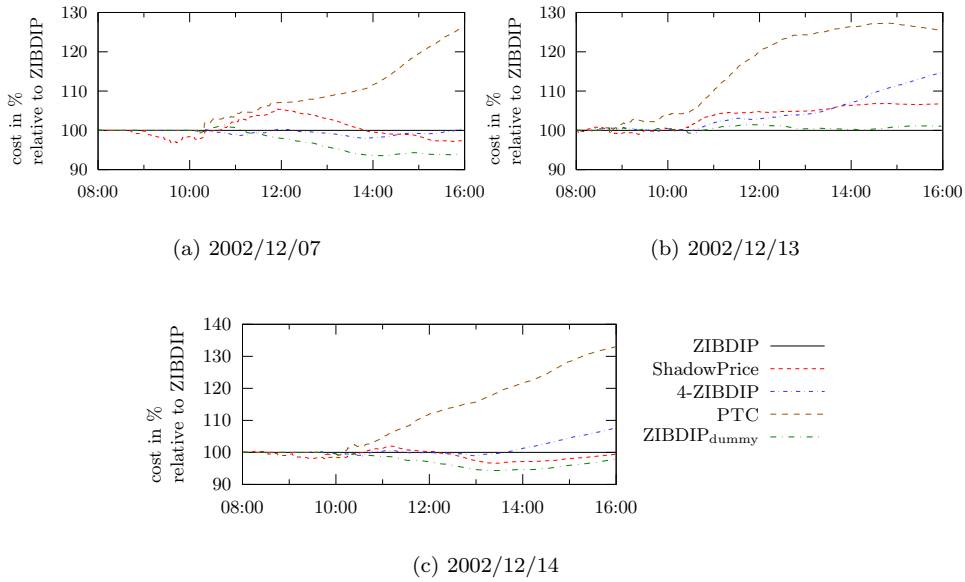
(c) PTC



(d) $\text{ZIBDIP}_{\text{dummy}}$

FIGURE 2. Optimality gaps and load ratios for simplified models and ZIBDIP. The optimality gap of ShadowPrice is inevitably infinite, since the lower bound the LP provides w. r. t. the modified cost (which is the reduced cost) is zero

ShadowPrice shows the worst L_1 norms, although the L_2 norms are good. We have no explanation for this.



(a) 2002/12/07

(b) 2002/12/13

(c) 2002/12/14

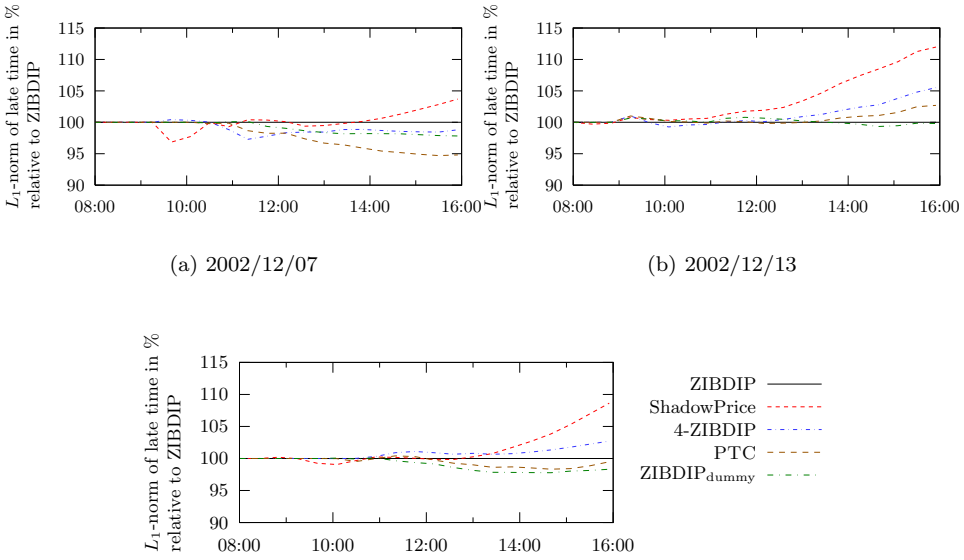
FIGURE 3. Comparison of ZIBDIP and simplified models w. r. t. the nonlinear cost function used by ADAC.

The good L_1 norms of PTC are due to the fact that, obviously, individial requests are postponed in favor of new requests that can be served faster. This can be seen very clearly in the L_2 norm diagrams, in which PTC performs worst. Uncontrolled deferment of requests is a very undesired property of an online algorithm. Therefore, PTC can not be recommended for tasks in which fairness is an issue. In our application, fairness certainly is an issue, whence the ADAC cost function contains a strictly convex waiting time penalty.

The answer to our main question is that the model error of most of our high-load models leads to worse long-term behaviour than the computational error that ZIBDIP produces (Fig. 3). Therefore, model simplifications have to be treated with great care. In our case, ZIBDIP$_{\text{dummy}}$ delivers the overall slightly best solution. One needs to be careful, though: a substantially smaller contractor delay of 45 min would lead to a tiny reoptimization gap; it, however, would at the same time produce unacceptable long-term costs because too many requests stay unassigned for too long. (This was, by the way, observed when we were looking for a good dummy contractor delay. Thus, ZIBDIP$_{\text{dummy}}$ involves some parameter tuning that the original ZIBDIP does not.)

4.2. **Simplified Reoptimization Algorithms.** The results so far could lead us to the conclusion to keep the orginal model but to use simplified reoptimization algorithms, since it seems that the optimality gap does not harm too much. After all, the implementation of a dynamic column generation procedure means a substantially larger effort, which is important especially in the industrial context.

Since we hear quite frequently such arguments in order to promote the use of heuristics rather than exact mathematical programming methods, we followed also



(a) 2002/12/07

(b) 2002/12/13

(c) 2002/12/14

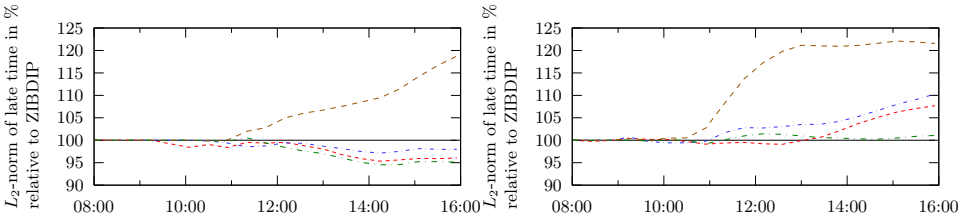
FIGURE 4. ZIBDIP vs. simplified models: L_1-norm of late time

this line in our simulation experiments and found out the following: Larger computational errors in the reoptimization can increase the long-term costs even more significantly than the model errors above.

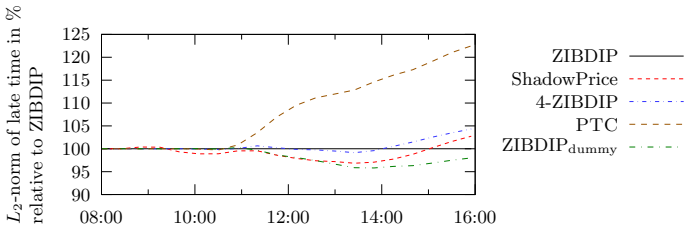
This is most incisively shown by the bad performance of BestInsert (Fig. 6, 7, and 8). Even 2-Exchange can not catch up with ZIBDIP and ZIBDIP$_{\text{dummy}}$ in the heavier instances. In the largest instance (b), 2-Exchange ends up at a long-term cost of 20 % above ZIBDIP and ZIBDIP$_{\text{dummy}}$. Especially striking is the fact that, in the large instance, the cost of 2-Exchange is constantly increasing over time relative to ZIBDIP. That means: the reoptimization errors accumulate.

In particular: in our application it is certainly not true, that deliberately sticking to the suboptimal solutions of heuristics like BestInsert in order to leave space for future requests can yield superior long-term behavior (compare [1, p. 5]). We are not saying that reoptimization is the best possible policy, maybe not even in our application. We claim: if anything is wrong with the reoptimization policy then this defect is not cured by using suboptimal solutions to the reoptimization problems.

The good overall performance of ZIBDIP$_{\text{dummy}}$ may stem not only from closing the optimality gap in the reoptimization process; it seems, moreover, that the special model of ZIBDIP$_{\text{dummy}}$ makes perfectly sense in the dynamic environment: since requests that are assigned to the dummy contractor would otherwise be served quite far in the future, with a high probability their position in the dispatch will change anyway. These considerations led us to the conclusion to install ZIBDIP$_{\text{dummy}}$ as the default reoptimization model in the automatic dispatching software for ADAC.

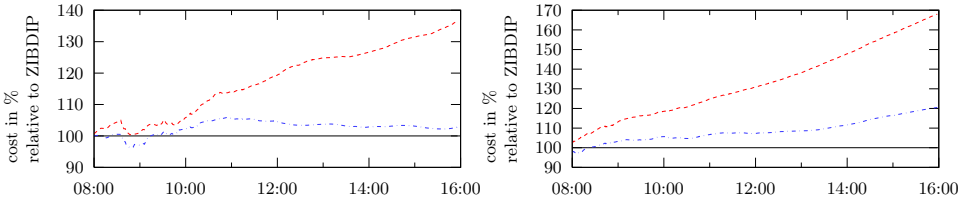


(a) 2002/12/07 (b) 2002/12/13

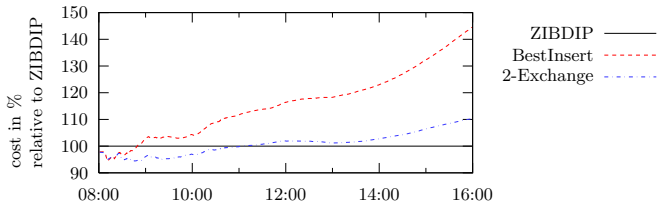


(c) 2002/12/14

FIGURE 5. ZIBDIP vs. simplified models: L_2-norm of late time

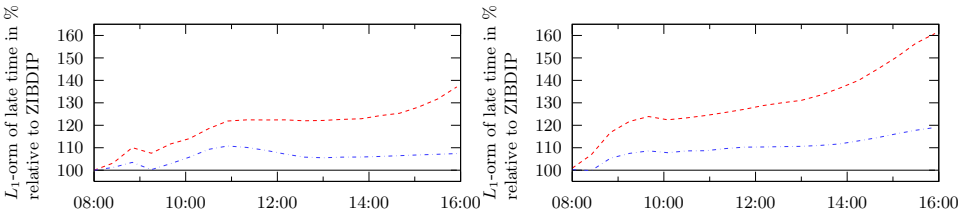


(a) 2002/12/07 (b) 2002/12/13

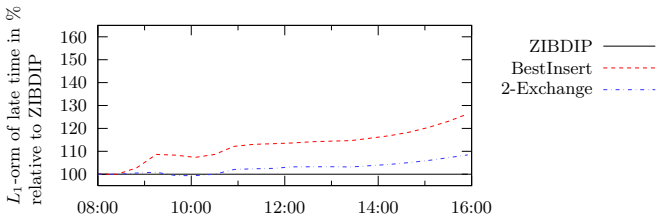


(c) 2002/12/14

FIGURE 6. Comparison of ZIBDIP and the heuristics w. r. t. the nonlinear cost function used by ADAC.

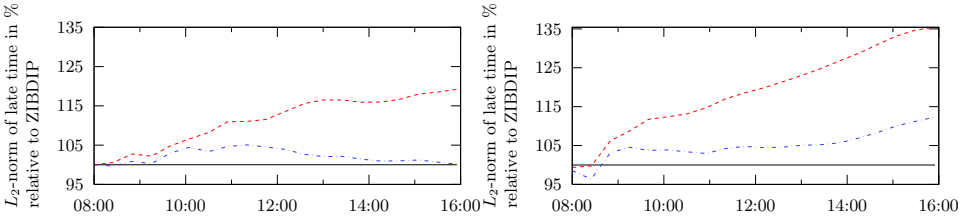


(a) 2002/12/07 (b) 2002/12/13

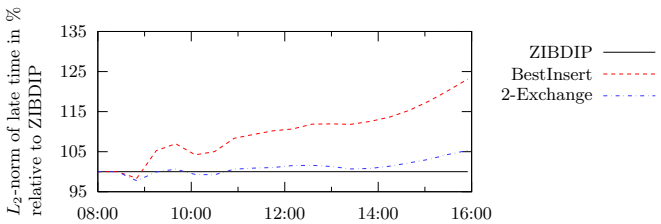


(c) 2002/12/14

FIGURE 7. ZIBDIP vs. heurisitics: L_1-norm of late time.



(a) 2002/12/07 (b) 2002/12/13



(c) 2002/12/14

FIGURE 8. ZIBDIP vs. heurisitics: L_2-norm of late time.

5. Significance

The production software for automated dispatching of ADAC service vehicles is delivered by Intergraph Public Saftety (IPS), based on the ZIBDIP algorithm. In the view of the results presented in this work, ADAC has filed a change request for the production software: $\text{ZIBDIP}_{\text{dummy}}$ is now the standard reoptimization model because it has proven to be more robust against sudden load increase. The key learning is that rigorous reoptimization on the basis of mathematical programming – though myopic w. r. t. unknown future requests – yields the best results in this particular application. Whether or not statistic information about future requests can be fruitfully ingetrated into the reoptimization framework, is work in progress.

References

[1] K. Q. Zhu, K.-L. Ong, A reactive method for real time dynamic vehicle routing problem, in: Proceedings of the 12th ICTAI, 2000.

[2] M. Grötschel, S. O. Krumke, J. Rambau, L. M. Torres, Online-dispatching of automobile service units, in: U. Leopold-Wildburger, F. Rendl, G. Wäscher (Eds.), Operations Research Proceedings, Springer, 2002, pp. 168–173.
URL http://www.zib.de/PaperWeb/abstracts/ZR-02-44/

[3] D. Bertsimas, D. Simchi-Levi, A new generation of vehicle routing research: robust algorithms, addressing uncertainty, Operations Research 44.

[4] S. O. Krumke, J. Rambau, L. M. Torres, Realtime-dispatching of guided and unguided automobile service units with soft time windows, in: R. H. Möhring, R. Raman (Eds.), Algorithms – ESA 2002, 10th Annual European Symposium, Rome, Italy, September 17–21, 2002, Proceedings, Vol. 2461 of Lecture Notes in Computer Science, Springer, 2002.
URL http://www.zib.de/PaperWeb/abstracts/ZR-01-22

E-mail address: krumke@mathematik.uni-kl.de
E-mail address: {hiller,rambau}@zib.de

{Benjamin Hiller, Jörg Rambau}, Department Optimization, Zuse-Institute Berlin, Germany

Sven O. Krumke, Department of Mathematics, University of Kaiserslautern, Germany