Combinatorial Online Optimization: Elevators & Yellow Angels*

Martin Grötschel, Benjamin Hiller**, and Andreas Tuchscherer

Konrad-Zuse-Zentrum für Informationstechnik Berlin, Department Optimization, Takustr. 7, 14195 Berlin, Germany.
E-mail: \{groetschel,hiller,tuchscherer\}@zib.de

1 Introduction

In *classical optimization* it is assumed that full information about the problem to be solved is given. This, in particular, includes that all data are at hand. The real world may not be so "nice" to optimizers. Some problem constraints may not be known, the data may be corrupted, or some data may not be available at the moments when decisions have to be made. The last issue is the subject of *online optimization* which will be addressed here. We explain some theory that has been developed to cope with such situations and provide examples from practice where unavailable information is not the result of bad data handling but an inevitable phenomenon.

We begin with some informal definitions concerning online optimization, refraining from giving formal definitions to avoid "technical overkill". Consider the following situation: we have to make immediate decisions, we have some data of the past on hand but information about future activities is unavailable, and we would like to make "good" decisions. In the classical optimization world we would study the problem to be considered, invent a mathematical model supposed to catch the key aspects of the problem, collect all data needed and use mathematical theory and algorithms to solve the problem. We call this approach in this paper *offline* from now on since the data collection phase is totally separated from the solution phase. In online optimization these phases are intertwined and there are many situations where this is not due to data acquisition difficulties but due to the nature of the problem itself.

A little more formal, in an *online problem*, data arrive in a sequence (that we will call *request sequence*), and each time when a new request arrives, a decision has to be made. If there are costs or some other objectives involved, we face an *online optimization problem* since we would like to make decisions in such a way that "at the end of the day" (when the last request has arrived and has been processed) the total cost is as small as possible.

There are lots of variants of this "basic online framework", and in each case one has to exactly state what the side constraints are, what online precisely means, how costs are calculated, etc. We outline a few of the possibilities that come up in practice.

For those familiar with *optimal control* the issues indicated here probably sound very familiar, and of course, online optimization is not a topic invented in combinatorial optimization or computer science. Online problems occur, for instance, when steel is cast continuously, chemical reactors are to be controlled, or a space shuttle reenters the atmosphere. There are some differences between these control problems and the *combinatorial online problems* we describe here. In the continuous control case, one usually has a mathematical model of the process. This is typically given in the form of a ordinary and/or partial differential equations. These describe the behavior of the real system under parameter changes over time. One precomputes a solution (often called optimal control) and the online algorithm has the task to adapt the running system in such a way that it follows the optimal trajectory. For instance, when a space craft reenters the atmosphere it is clear where it has to land. It should follow a predetermined trajectory to the landing place. The measuring instruments determine the deviations from this trajectory, and the algorithms steering the space craft make sure that the craft follows the trajectory using the available control mechanisms on board.

In contrast, the combinatorial online optimization problems we consider have no "trajectory" that one could precompute. Decisions are *discrete* (yes or no), can only be made at certain points in time and not continuously, and in most cases, decisions once made cannot be revoked. In other words, although optimal control and

* Supported by the DFG Research Center Matheon *Mathematics for key technologies* in Berlin.
** Supported by the DFG research group "Algorithms, Structure, Randomness" (Grant number GR 883/10-3, GR 883/10-4).

combinatorial online optimization have certain aspects in common, there are significant differences that lead to different problems in practice and require different mathematical treatment.

Let us look at an example. Recall your last move to a new apartment. Moving the furniture into a truck and packing them safely is a nontrivial task. From a mathematical point of view we call this a three dimensional packing problem. Cutting the pieces of a suit from a cloth roll to minimize waste is a similarly demanding task. An extremely simplified version of these two problems is the *bin packing problem*. We have a (possibly infinite) number of one dimensional bins, all of the same height. Certain items (all with the same "footprint", matching the footprint of the bins, but possibly with different heights) have to be packed in the bins. The goal is to use as few bins as possible. If all items are given in the beginning, we have an offline bin packing problem. If items are generated one by one and we have to pack each item as soon as it arrives and if we are not allowed to repack, we have a "standard" online bin packing problem. It may be that we are allowed to keep only a fixed number of bins, say q bins, open and that we are also allowed to move items between the open bins. But as soon as we need a new bin, we must close one bin and move it away. We call this problem the *q-restricted online bin packing problem with repacking*.

The latter is a typical situation that comes up when you move. A certain number of boxes are open in a room, you pack items into the boxes, are allowed to repack, but due to lack of space, whenever a new box is needed, one currently open box has to be closed and moved away. Again, the goal is to use as few boxes as possible.

There may also be some clock running. Requests appear in sequence and you are not necessarily obliged to serve them immediately (as in the standard problem above), but there may be extra costs incurred with every time unit a request that has appeared is not served. This version of an online optimization problem will be called the *time stamp model*. A natural application is the ADAC Yellow Angels problem to be discussed later.

Given an online optimization problem (e. g., one of the versions described above) then an *online algorithm* is an algorithm that makes decisions respecting the side constraints of the particular online problem that it is designed for. In the standard model of online computation the algorithm has to decide immediately what to do and may not reconsider the decision later (e. g., it packs an item into a bin and cannot repack). In case of the q-restricted online bin packing problem with repacking the online algorithm may repack the q open bins and move items between them. But as soon as a bin is closed the algorithm has no further access to the items in the closed bin. Similarly, in the time stamp model the online algorithm is not always obliged to act immediately at the appearance of a request, but may have to pay a lot if service is delayed.

So, what an online algorithm is allowed or supposed to do depends on the special circumstances and the side constraints. There is another issue: time restrictions. For online algorithms in the standard (or repacking) model running time of an algorithm is not an issue. We just do not care (in theory). These algorithms may use any amount of computing time or computing power. The issue of importance here is the lack of information about the future.

If time is precious we speak of *real time problems* and *real time algorithms*. Again, what real time means depends crucially on the time frame of the process considered. Routing decisions in telecommunication (an important real time problem) have to be made almost immediately, while in the Yellow Angels case, ten seconds are an upper bound for the online algorithm to come up with the decision. Control algorithms for elevators may use about a second for a decision, but decision times in the range of minutes would not be tolerated by the customers.

An area where online problems occur in abundance is logistics which is at the heart of modern production and service facilities. In many real-world applications logistics systems are automatically supervised and permanently controlled. The online problems arising here usually correspond to the time stamp model (occasionally with short time preview) with additional real-time requirements.

As an example, consider an automated warehouse used to store and retrieve goods. Pallets with goods have to be stored and are to be transported from the storage to vehicles for further distribution. The overall goal of a good control is to ensure a constant flow of pallets such that the vehicles do not have to wait too long, to avoid congestion, and to coordinate the traffic. The control decisions are usually discrete, e. g., which pallet to route on which conveyor belt and/or vertical elevator.

This control problem is again online since control decisions have to be made in view of an unknown future and the control has to be updated each time new information becomes known. this is called an *online algorithm*. The time frame for the online algorithms in this area is usually in the range of milliseconds to seconds – depending on the mechanical speed of the stacker cranes or other moving devices. In order to develop good online algorithms it is important to analyze the impact of the control decisions on the current and future performance of the system.

2 Theoretical Analysis of Online Optimization Problems

In this section we describe the approach usually used for evaluating the quality of a given online algorithm. This theoretical concept is called *competitive analysis*. Before introducing competitive analysis in detail, we present some example problems that are revisited later on.

2.1 Examples for Online Optimization Problems

Online Traveling Salesman Problem.

The probably most famous combinatorial (offline) optimization problem is the Traveling Salesman Problem or TSP for short. An instance consists of a set of locations that are to be visited in such a way that the total distance traveled is minimized.

To define the online version of the TSP in detail, we need to introduce the notion of a metric space. A metric space is simply an abstract way to "measure" distances. It consists of a set of *points* M and a function $d\colon M \times M \to \mathbb{R}_{\geq 0}$) supposed to give distances between each pair of points. Therefore, d has to satisfy the following properties:

1. $d(u,u) = 0$ for all $u \in M$
2. $d(u,v) = d(v,u)$ for all $u, v \in M$ (symmetry)
3. $d(u,v) + d(v,w) \geq d(u,w)$ for all $u, v, w \in M$. (triangle inequality)

The online version of the TSP is as follows. We are given a metric space (M,d) with a distinguished point o, the *origin*, where the "server" is located at time 0, and a sequence of requests $r_1, r_2, \ldots$. Each request is a pair $r_j = (t_j, p_j)$ specifying the time $t_j \geq 0$ at which r_j is released and a point $p_j \in M$. The release times form an ordered sequence in the sense that $t_i \leq t_j$ if $i < j$. The server does not move faster than unit speed. This problem is called Online Traveling Salesman Problem, for short OnlineTSP.

There are several reasonable objectives for the OnlineTSP, e. g.:

Completion time: This is the total time for serving all requests and returning to the origin o.
Maximum waiting time: The waiting time of a request r_j is the difference between the time when the r_j is served (the server reaches the point p_j) and the release time t_j.
Average waiting time: The average of all waiting times.

There are cases where more than one of these objective functions need to be considered. This leads us to multicriteria online optimization, a subject that has received almost no attention in the research literature. We will touch upon the issue of "balancing" the objective functions above in the elevator case study in the next section.

Online Bin Coloring.

Another simple online optimization problem is Online Bin Coloring, which is as follows. We are given a request sequence consisting of unit size items $r_1, r_2, \ldots$ where each item has a color $c_j \in \mathbb{N}$. The items are to be packed into bins, all of the same size $B \in \mathbb{N}$, as soon as they arrive. Repacking an item later on is not allowed. At each moment there are $q \in \mathbb{N}$ empty or partially filled bins. Whenever a bin is full, it is closed and replaced by a new empty bin. The objective is to pack the items in such a way that the maximum number of different colors in a bin is as small as possible.

2.2 Competitive Analysis

Competitive analysis provides a framework to measure theoretically the quality of online algorithms. More precisely, it seeks to answer the question what is lost in the worst-case due to lack of information. Competitive analysis was introduced formally in 1985 by Sleator and Tarjan in [15]. The term competitive analysis was first used in [10]. However, Graham already used this method in 1966 for evaluating algorithms for machine scheduling, see [6]. For an overview on competitive analysis, we refer to [2].

In the following, we only consider online minimization problems. Competitive analysis compares the performance of a given online algorithm ALG to that of an algorithm that knows the complete request sequence in advance and can serve the requests at a minimum cost. We call this benchmark algorithm *optimal offline algorithm* OPT. For a request sequence σ, we denote the corresponding optimal offline cost and the cost of ALG by $\mathrm{OPT}(\sigma)$ and $\mathrm{ALG}(\sigma)$, respectively. The algorithm ALG is said to be c-competitive for $c \in \mathbb{R}, c \geq 1$ if

$$\mathrm{ALG}(\sigma) \leq c \cdot \mathrm{OPT}(\sigma)$$

for all request sequences σ. The *competitive ratio* of ALG is the smallest value c such that ALG is c-competitive. ALG is called *competitive* if ALG is c-competitive for some $c \geq 1$.

An online algorithm is *competitive*, if it competes "well" with the optimal offline algorithm on *all* request sequences. That is, competitive analysis is a *worst-case measure*. Hence, in order to show that the competitive ratio of an online algorithm is bad, it suffices to find one sequence of requests on which the online algorithm appears bad compared to the optimal offline algorithm.

We can think of competitive analysis as a game between an *online player* and a *malicious adversary*. The adversary constructs a sequence of requests to be processed by the online player. The adversary has complete knowledge of the online player's strategy that corresponds to an online algorithm, and he intends to construct a sequence such that the cost ratio for the online player and the optimal offline cost is maximized.

Examples.

Recall the standard online bin packing problem as described in the introduction. The algorithm FirstFit which packs each item in the first open bin providing sufficient space is known to be 1.7-competitive. Van Vliet [18] proved that no online bin packing algorithm can be better than 1.5401-competitive. There are many algorithms known that achieve a better competitive ratio than FirstFit, but none attains the lower bound.

The situation is different for the q-restricted online bin packing with repacking. Lee and Lee [14] showed a lower bound of 1.69103 for the competitive ratio of every online algorithm. Galambos and Woeginger [5] gave an algorithm that achieves this competitive ratio and is thus optimal. It is interesting to note that the advantage gained by allowing repacking is more than outweighed by the restriction of using at most q bins.

Next we present some surprising competitive analysis results obtained for OnlineTSP and Online Bin Coloring.

Consider an algorithm ALG for the OnlineTSP problem on one of the most trivial metric spaces one can think of, the one-dimensional line of real numbers. The server is supposed to start in the origin and we want to minimize the maximum waiting time of a request. We want to be a nasty adversary and to achieve a high ratio between the optimal offline cost and that of ALG and do the following. At time 0 ALG has to decide what the server does. There are three cases: Either the server remains at the origin, it moves to the left or to the right. Suppose the server travels to the right. Then the first request in the sequence arrives at $x = -1$ at time 1. Since the server moved to the right, it will not arrive at the request at time 1 or earlier, so it gets a positive waiting time. The offline algorithm, however, knows where the request will occur. So he can move the server to the position -1 on the line before the request occurs achieving a waiting time of 0. This looks like a "dirty trick", but we have thus shown that no online algorithm can achieve a constant competitive ratio. This essentially means that all online algorithms are equally bad from the viewpoint of competitive analysis.

Let us now turn to Online Bin Coloring. A "natural" algorithm would put an item with a color already present in one of the bins into the same bin. If the color is currently not present in one of the bins, one would put it in a bin which currently has the least number of distinct colors. Let us call this algorithm GreedyFit. It can be seen [11] that GreedyFit achieves a competitive ratio larger or equal to $2q$ (q is the number of bins that can be open simultaneously). The totally stupid algorithm OneBin, which uses only one bin until it is filled completely and puts all items into that bin, achieves a competitive ratio of at most $2q - 1$, making it superior to GreedyFit in terms of the competitive ratio.

Of course, there are many online problems where competitive analysis yields useful results, but for those two problems arising in logistics the results are not very helpful.

Extensions of Competitive Analysis.

In the last section we have seen examples where competitive analysis yields surprising results. In various online optimization problems the following phenomena arise:

- There does not exist a competitive online algorithm at all.
- All online algorithms have the same competitive ratio.
- A reasonable online algorithm that performs well in simulation experiments has a worse competitive ratio than an algorithm that obviously performs bad in practice.

The reason for these phenomena is the worst-case nature of competitive analysis. In other words, the malicious adversary is "too strong". Various concepts to reduce the power of the adversary have been proposed in the literature. The most direct way is to restrict the request sequences that the adversary can produce in some reasonable way or to limit its power directly. For instance, the trick used for the OnlineTSP would not work anymore if the adversary is not allowed to travel in a direction where no request is known at present.

Another possibility is to consider *randomized online algorithms*, which are allowed to use random decisions for processing requests. The cost of a randomized algorithm is a random variable, and we are interested in its

expectation. Normally, the malicious adversary knows the distribution used by the online player but cannot see the actual outcome of the random experiments. As a consequence, he must choose the complete input sequence in advance. This type of adversary is called *oblivious adversary* (see [2] on different types of adversaries). Yet another alternative is to consider random request sequences and doing *average case analysis*, but this requires some probabilistic assumptions on how the inputs looks like.

Theoretical analysis can give important insights in possible weaknesses of online algorithms and how to overcome them. However, to really evaluate how an algorithm performs in practice it is often necessary to resort to a simulation of the system. It is certainly no insult to state that the theory developed for online optimization rarely provides good guidelines for the choice of online algorithms to be employed in practice.

3 Case Study: Elevator Systems

Online problems arise frequently in logistics applications, for instance in high rack warehouses. One particular system was studied in [7], dealing with the distribution center of Herlitz PBS in Falkensee near Berlin. This distribution center stores and retrieves pallets using a complex system of conveyor belts, elevators, and other transportation devices. The overall goal for the control is to ensure a rapid, congestion-free flow of pallets through the system. The pallets to be transported on a production day are not known in advance, which makes this problem an online problem.

In the following, we concentrate on the control of the elevators, which constitute an important subsystem. In the Herlitz distribution center, two groups of elevators work together and it is crucial for the system performance to coordinate each group of elevators well.

Elevator control problems can be seen as a generalization of the OnlineTSP problem, where a request consists of two points instead of one, with the interpretation that the server needs to travel from the first to the second point in order to serve the request. In addition, there is not only one server, but possibly many. Such problems are known as *Dial-a-Ride problems (DARP)*. We call the online version OnlineDARP. The time a request spends in the system is called the *flow time*, consisting of the waiting time of the request and the time needed to serve it.

3.1 Theoretical results

Various theoretical results for OnlineDARP have been established. For the completion time objective, there is a 2-competitive OnlineDARP algorithm which was proposed in [1], which means that this online algorithm may take twice as long as the best possible solution. The authors also show that no algorithm can be better than 2-competitive, in other words, whatever smart algorithmic idea someone may have, there is a request sequence on which the online algorithm takes twice as long as the best solution.

The same paper also analyzes two general online strategies, REPLAN and IGNORE. The REPLAN strategy computes a new optimal schedule each time a new request becomes known. This is often used in practice. The IGNORE strategy works in phases. At the beginning of each phase, an optimal schedule for the currently known requests is computed and realized. Only when the schedule has been finished a new schedule for the then known requests is computed and a new phase starts. During execution of a schedule all new requests are ignored, which gave the strategy its name. Both REPLAN and IGNORE are 5/2-competitive [1] and thus not best possible.

We already saw that no online algorithm can be competitive when minimizing the maximum flow time in the case of the OnlineTSP. Since OnlineTSP is a special case of OnlineDARP, this holds for OnlineDARP as well. For the OnlineTSP on the real line there is an online algorithm which is 8-competitive against a *non-abusive* adversary [12], which is a restricted kind of adversary. For a reasonably restricted class of request sequences, the IGNORE strategy achieves bounded maximum and average flow times [8] which is interesting since this guarantees some kind of stability of the system.

3.2 Simulation results

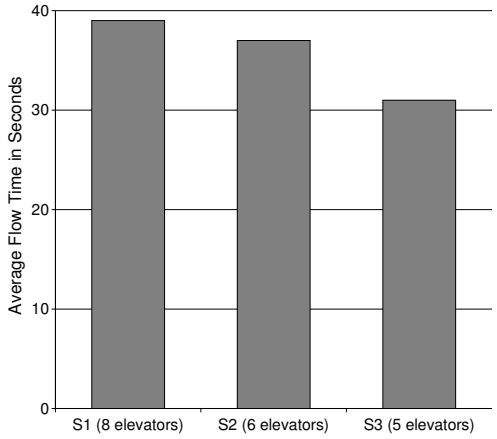
The elevator control problem has been studied in simulations, too. The particular setting of the Herlitz distribution center was investigated in [7], which compared seven algorithms. It turned out that the algorithms had quite a different performance with respect to average and maximum flow time and that the algorithms used in production were significantly inferior. Another observation was that algorithms achieving good average flow times give high maximum flow times and vice versa. The REPLAN algorithm achieved very good average flow times, but also very high maximum waiting times. The IGNORE algorithm, on the other hand, gave

worse but still acceptable average flow times, but good maximum flow times as well. A variant of the IGNORE algorithm that balances the two objectives was therefore recommended for implementation at the plant. This is the way how we treated the multicriteria aspect of our Herlitz elevator problem. We looked for an algorithm that produces reasonable values for all relevant objectives.

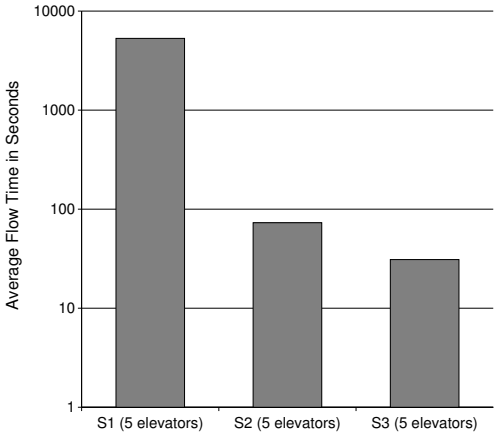
The algorithms considered in [7] were all quite simple from a methodological point of view. More recently, Friese and Rambau [4] studied a more involved algorithm based on Mathematical Programming. It is a modified version of the algorithm ZIBDIP which was developed for routing service vehicles (see next section for more on this).

Among other algorithms, they compared the modified version of ZIBDIP called REOPT to the algorithms FIFO and NN ("NearestNeighbor"), each representing a certain class of typical algorithms. FIFO assigns a new request in round-robin fashion to the elevators, serving each request of an elevator in the order of arrival. FIFO is an example of a simple rule-based algorithm. NN assigns a new request to the elevator giving the lowest waiting time for the new request if it is inserted such that no other request is postponed. The requests of each elevator are served such that the distance from the last to the next request is minimized, hence the name of the algorithm. NearestNeighbor represents a simple greedy heuristic frequently used in practice. REOPT, finally, determines the schedule in an integrated way (the assignment and service order decisions are taken into account simultaneously) such that the average waiting time of all the requests is minimized.

Figure 1 displays comparisons of these algorithms. Figure 1(a) shows the number of elevators needed to get an average flow time of at most 40 seconds for the given set of requests and the average flow time actually achieved with this number of elevators. It turns out that REOPT achieves this performance with five elevators only, whereas FIFO and NN require eight and six elevators, respectively. The average flow times achieved by the algorithms when they use five elevators only are shown in Figure 1(b). Note that the vertical axis has a logarithmic scale! FIFO, in fact, turns out to be an unstable elevator scheduling system overflowing the buffer space enormously. FIFO is much worse than NN and REOPT. Although NN is able to handle the traffic without overflowing the system, REOPT outperforms NN significantly.



(a) Number of elevators needed to serve a 16 floor system with average flow time at most 40 seconds for FIFO (S1), NN (S2), and REOPT (S3)



(b) Average flow time achieved by FIFO (S1), NN (S2), and REOPT (S3) on a 5 elevator 16 floor system. Note that the y-axis is logarithmic.

Fig. 1. Results of comparison of elevator control algorithms obtained in [4].

The conclusion of [4] is that multi-elevator systems with capacity 1 can be efficiently controlled by sophisticated online algorithms.

It will be interesting to see whether similar results can be obtained for the control of passenger elevators. Passenger elevators present other kinds of challenges. First of all, the information available to the elevator control is much more limited, since passengers specify only their travel direction (up, down) when they forward their request. In contrast, for industrial cargo elevator systems the destination is known, which allows much better control. In recent years some elevator companies started to implement destination call control for passenger elevators, too. The passenger is now required to enter the destination floor instead of the direction only and it is claimed that this additional information leads to an increased performance. We are currently investigating such elevator systems.

Even in destination call systems there are other features distinguishing passenger elevators from cargo elevator systems. The cabin of a passenger elevator usually has a relatively large capacity. This in conjunction with additional requirements such as that no passenger travels temporarily in the wrong direction makes the schedules more complex. But even here Mathematical Programming methods can provide powerful elevator control algorithms [16, 17].

4 Case Study: ADAC Yellow Angels

The ADAC is the German Automobile Association. It operates a fleet of service vehicles, known as "Yellow Angels", providing help to motorists who had a breakdown. Help requests are registered in one of five help centers, which coordinate the service vehicles that are on duty. Each of the vehicles is equipped with a GPS system that reports to the help centers precise information on the current location of the vehicles. The dispatchers may employ, in addition to ADAC's Yellow Angels, service vehicles from contractors. The overall goal of the planning is to provide a high quality service at low operational cost, two objectives that obviously cannot be optimized simultaneously. This implies that "rules of compromise" have to be found. Good quality of service in our case means short waiting times for the motorists, where the meaning of "short" depends on the system load.

The whole fleet of ADAC service vehicles consists of more than 1600 heterogeneous vehicles. Moreover, ADAC's contractors have some 5000 vehicles that may be employed by the ADAC help centers. In high load situations, there are more than 1000 help requests per hour. These numbers give a rough indication about the large-scale nature of the problem.

This vehicle scheduling problem is a prototypical online problem, since in order to provide a continuous operation the schedule needs to be updated every time a new help request enters the system. ADAC wanted to establish an automatic planning system that provides good vehicle schedules in about 10 seconds computation time to guarantee real-time operation.

From a theoretical point of view, this problem is very similar to the OnlineTSP problem. The main differences are that there are many servers (up to 200 service vehicles per help center) and that the server needs to spend some time at every help request (equivalent to a city visit). For the practical design of online algorithms the theory known so far does not help much, since there are no competitive algorithms known for higher dimensional metric spaces than the real line.

ADAC's requirements suggested to build an online algorithm based on the *reoptimization approach*: Every time some new information becomes available, a *snapshot problem* is constructed that reflects the current state of the system. This snapshot problem is then solved, giving a new schedule that is followed until the schedule is updated again.

More precisely, the snapshot problem consists of the following data:

- location, time of occurrence and estimate of service time of each help request (henceforth called *requests*),
- current position, home position and service status of each vehicle (henceforth called *unit*),
- set of available contractors,
- information about operational cost of units and contractors,
- information about the capabilities of each unit, i. e., what kind of help request can be handled by that unit.

The goal is to compute a good schedule servicing all the requests. Here a schedule is a set of tours for each vehicle that is on duty. But what is "good" schedule? The goals are high quality of service and low operational cost so this problem is in fact a multi-criteria optimization problem. Clearly, these two goals are partially contradicting. To cope with both criteria they are combined by defining the overall cost as a weighted sum of penalty cost for bad service quality and the operational cost. Of course, suitable weights reflecting the relative importance need to be determined for this approach. Alternatively, and more fundamentally one could try to compute further solutions from the Pareto set. This, however, is very difficult to accomplish in one short period of time (seconds) allowed in this application.

It is simply impossible to take all aspects of this dispatching problem into account. Yellow Angels customers may ask for special treatment, e. g., a seriously sick person is in the car, an important manager needs to get to an airport or a pregnant woman to a hospital, etc. We simply do not know how to rank such requests in general. Therefore, only the major components of the Yellow Angels dispatching problem (time and cost) are modeled mathematically and considered in the algorithm, while "special requirements" are left to the human decision maker to be treated in "post processing".

This ADAC snapshot problem is a typical *Vehicle Routing Problem* (VRP) and the solution approach sketched in the following is often used to solve VRPsTo keep things a bit simpler, we ignore the contractors

from now on; they can be integrated quite easily. The following method was proposed in [13] and has been implemented in ADAC's service scheduling system.

A *tour* t_u for a unit is a sequence of requests, each of which can be handled by the unit. The tour t_u gives the order in which the requests are to be served, which in turn allows to compute the (estimated) waiting times for each customer. This way, a cost value $c(t_u)$ can be associated to the tour, which consists of the "service quality cost" and the operational cost incurred by the tour. For the computation of the cost it is assumed that the unit travels to its home position after finishing the last request of the tour.

For each unit u, let $\mathcal{T}_u$ be the set of all tours for that unit and $\mathcal{T}$ be the union of all the sets $\mathcal{T}_u$. A schedule is now a selection of one tour for each unit such that all the requests are contained in exactly one tour. This selection can be modeled by introducing a binary variable x_t for $t \in \mathcal{T}$; $x_t = 1$ means that the tour is selected. An optimal schedule can now be computed by solving the following integer program (IP).

$$\min \sum_{t \in \mathcal{T}} c(t) x_t$$

$$\sum_{t \in \mathcal{T} :\, r \in t} x_t = 1 \qquad \forall r \in R \tag{1}$$

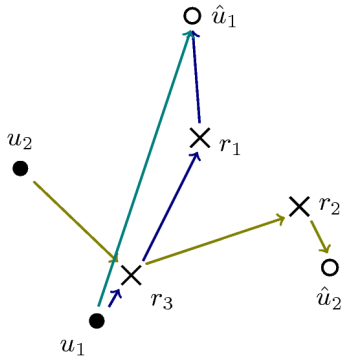
$$\sum_{t \in \mathcal{T}_u} x_t = 1 \qquad \forall u \in U \tag{2}$$

$$x_t \in \{0,1\} \quad \forall t \in \mathcal{T} \tag{3}$$

The constraints (1) ensure that each request is contained in exactly one tour, whereas due to constraints (2) exactly one tour for each unit is selected. Figure 2 illustrates the idea of this IP model. In more abstract terms, we can write the IP as

$$\min c^T x$$
$$Ax = 1$$
$$x \in \{0,1\}^n$$

where the binary matrix A and the cost vector c is constructed as depicted in Figure 2(b). An Integer Program of this type is known as a *set partitioning IP*.



(a) Some tours for two units and three requests r_1, r_2, r_3. u_1 and u_2 indicate the current positions of both units and $\hat{u}_1$ and $\hat{u}_2$ their home positions.

	t_1	t_2	t_3	...	t_N	
	1	0	0	...		r_1
	0	1	0	...		r_2
A	1	1	0	...		r_3
	1	0	1	...		u_1
	0	1	0	...		u_2
c^T	c_{t_1}	c_{t_2}	c_{t_3}	...	c_{t_N}	
x^T	x_{t_1}	x_{t_2}	x_{t_3}	...	x_{t_N}	

(b) Representation of the tours in Figure 2(a) as entries of a matrix A and a corresponding cost vector c.

Fig. 2. Illustration of the tour-based model (1)–(3).

One problem with this set partitioning IP is that is has an enormous number of columns as there are extremely many possible tours. However, a technique known as *column generation* [3] in Mathematical Programming is applicable in this case due to the special structure of the constraint matrix. Column generation allows the implicit treatment of all the columns of the matrix without explicitly constructing all of them. This is done by searching for improving tours via the solution of an auxiliary problem, which in this case turns out to be a *resource constraint shortest path problem (RCSP)*.

Both the set partitioning problem and the RCSP are known to be NP-hard combinatorial optimization problems. This means that solving them to optimality is usually quite time-consuming, even if advanced

methods are used. In the context of our application for the ADAC, however, the schedule needs to be computed very fast since only 10 seconds computation time are allowed.

A careful adaptation of the solution algorithm to the problem structure facilitates finding good and even verifiably optimal solutions in the allowed computation time. One particular feature of the problem is that the cost corresponding to the waiting time of the requests make long tours expensive, so the tours in the final schedule are usually short (at most 5 or 6 requests in high load situations). This allows to solve the RCSP by a controlled implicit enumeration of the tours. The set partitioning problem is usually easier to solve if the sets are small. Moreover, since requests that cannot conveniently be covered by a unit can be outsourced to a contractor it is always easy to complete a partial covering by the units to a complete covering of the requests. Therefore, good integer solutions (i. e., schedules) are obtained very early in the solution process. These considerations lead to the development of the algorithm called ZIBDIP.

To give an impression on what is achieved by ZIBDIP, we cite some results. In a first study, ZIBDIP was compared to some variants of a genetic algorithm suggested by a software vendor [13]. The genetic algorithm GA could be initialized using two different starting heuristics IT and CL as well as a best-insertion heuristic HEU. The averaged results for three high load snapshot problems are given in Figure 3. All the genetic algorithm variants finish with solutions with costs that are more than 15% above the optimal value after 15 seconds computation and are still more than 5% away even after 120 seconds. ZIBDIP, in contrast, is within 1% of the optimal value already after 5 seconds, showing the fast convergence of this method needed for real-time application.

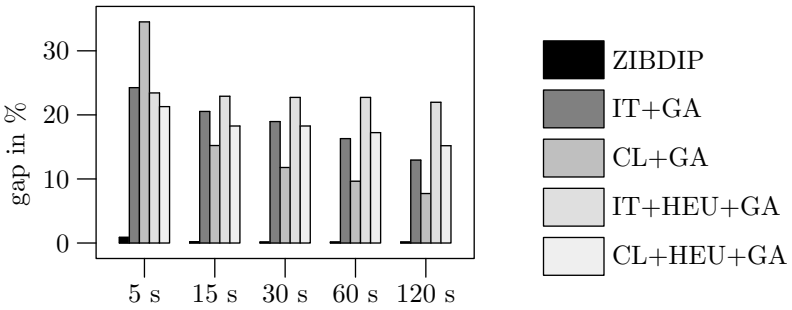


Fig. 3. Comparison of ZIBDIP with several variants of genetic algorithms. Depicted is the deviation from the optimal cost in percent after a given computation time. The data is taken from [13].

Figure 4 shows the quality of the solutions to the snapshot problems for one day obtained by ZIBDIP. Each point specifies a guarantee on the solution quality w. r. t. the optimal snapshot cost. This deviation is called *optimality gap*. In addition, the figure shows the *load ratio*, which is the ratio of requests and unit in the snapshot problem. The optimality gap gets worse for higher load ratios, but does not exceed 7% on the whole day.

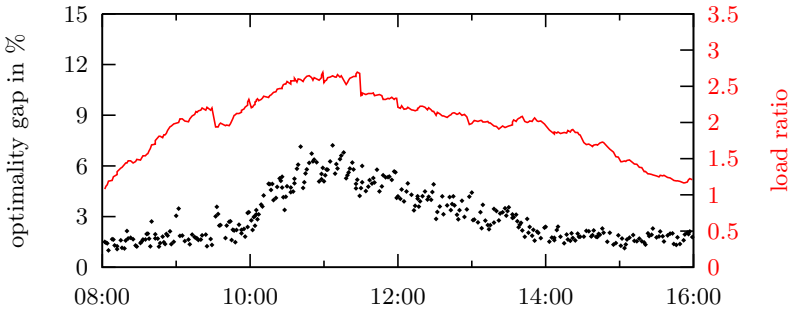
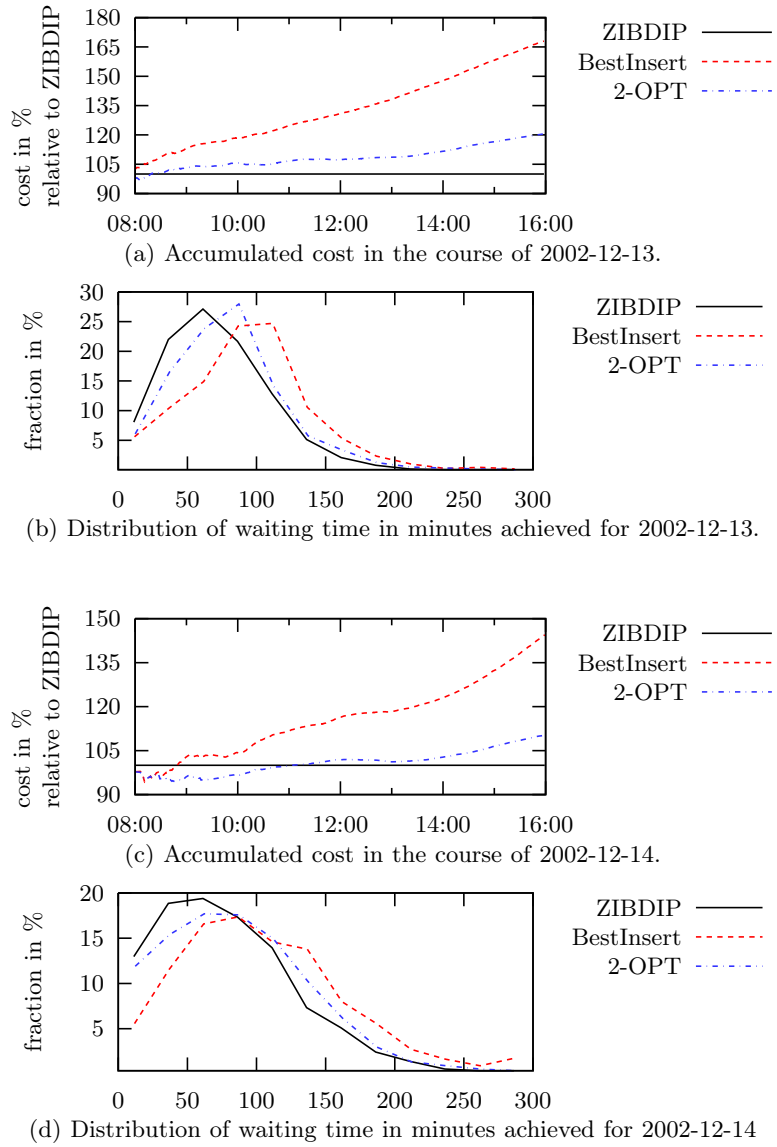


Fig. 4. Solution quality in the course of a high load day [9].

Finally, Figure 5 compares the performance of ZIBDIP with those of two relatively simple heuristics. The first one, BestInsert, builds a new schedule by inserting new requests in the existing schedule such that the cost increases as little as possible. The second heuristic, 2-OPT, computes an initial solution via BestInsert and then iteratively tries to exchange requests in one tour or between two tours in order to decrease the cost until no improvement is possible. In the long run, ZIBDIP produces overall schedules that are 40-60% “cheaper”

than that of BestInsert and 8-20% "cheaper" than that of 2-OPT. The improvement in the waiting times is significant, too: ZIBDIP serves a larger share of requests with lower waiting time than the two heuristics.



(a) Accumulated cost in the course of 2002-12-13.

(b) Distribution of waiting time in minutes achieved for 2002-12-13.

(c) Accumulated cost in the course of 2002-12-14.

(d) Distribution of waiting time in minutes achieved for 2002-12-14

Fig. 5. Comparison of ZIBDIP to two heuristics on two high load days [9].

All in all, ZIBDIP is a powerful algorithm and another example that Mathematical Programming methods can lead to real-time compliant online algorithms and codes.

5 Conclusion

Combinatorial online optimization problems, arising in structure generation under varying goals. appear in a wide range of practical applications. In most "industrial environments" we have had access to they are treated in a "very heuristic" manner, i. e., ad hoc algorithms are employed. Moreover, these algorithms are often programmed by persons who have no experience concerning all the possible pitfalls such heuristics may run into.

The state of the theory of online optimization is not in best possible shape either. There are various concepts of analysis, such as competitiveness, but it is questionable whether competitiveness provides good guidelines for practice.

At present, it seems that individual case analysis consisting of both, theory and simulation, is necessary. In the examples we outlined in this section we have gained stimuli for theoretical studies from simulation

experiments, and theoretical investigations guided the design of the algorithms that were finally employed in practice. Overall, serious mathematical analysis and the careful algorithms design, based on the many techniques offered by mathematical programming, pays significantly. Our examples showed that these efforts lead to much more robust online systems with significantly better performance.

Is there nothing left to be desired? Of course not. We would wish to have a better understanding of the methodology to cope with online problems. The success stories reported here are based on very significant research efforts, all very problem specific and not really guided by "general principles" that would result from a good theoretical underpinning. "Online" is not well understood. Moreover, online problems often have more than one objective – as indicated before. Multicriteria online optimization, however, does currently not exist as a research field. It is, at best, in its infancy.

References

1. Norbert Ascheuer, Sven O. Krumke, and Jörg Rambau, *Online Dial-a-Ride problems: Minimizing the completion time*, Proceedings of the 17st Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Computer Science, vol. 1770, Springer, 2000, pp. 639–650.
2. Allen Borodin and Ran El-Yaniv, *Online computation and competitive analysis*, Cambridge University Press, 1998.
3. Guy Desaulniers, Jacques Desrosiers, and Marius M. Solomon (eds.), *Column generation*, GERAD 25^{th} anniversary series, Springer, 2005.
4. Philipp Friese and Jörg Rambau, *Online-optimization of a multi-elevator transport system with reoptimization algorithms based on set-partitioning models*, Discrete Appl. Math. **154** (2006), no. 13, 1908–1931, also available as ZIB Report 05-03.
5. G. Galambos and G. J. Woeginger, *Repacking helps in bounded space on-line bin-packing*, Computing **49** (1993), 329–338.
6. Ronald L. Graham, *Bounds for certain multiprocessing anomalies*, Bell System Technical Journal **45** (1966), 1563–1581.
7. Martin Grötschel, Dietrich Hauptmeier Sven O. Krumke, and Jörg Rambau, *Simulation studies for the online Dial-a-Ride problem*, Tech. Report SC 99–09, Zuse Institute Berlin, 1999.
8. Dietrich Hauptmeier, Sven O. Krumke, and Jörg Rambau, *The online Dial-a-Ride problem under reasonable load*, CIAC 2000, Lecture Notes in Computer Science, vol. 1767, Springer, 2000, pp. 125–136.
9. Benjamin Hiller, Sven Oliver Krumke, and Jörg Rambau, *Reoptimization gaps versus model errors in online-dispatching of service units for ADAC*, Discrete Appl. Math. **154** (2006), no. 13, 1897–1907, Traces of the Latin American Conference on Combinatorics, Graphs and Applications – A selection of papers from LACGA 2004, Santiago, Chile.
10. Anna Karlin, Mark Manasse, Larry Rudolph, and Daniel Dominic Sleator, *Competitive snoopy caching*, Algorithmica **3** (1988), no. 1, 79–119.
11. Sven Oliver Krumke, Willem E. de Paepe, Leen Stougie, and Jörg Rambau, *Online bin coloring*, Proceedings of the 9th Annual European Symposium on Algorithms (Friedhelm Meyer auf der Heide, ed.), Lecture Notes in Computer Science, vol. 2161, 2001, pp. 74–84.
12. Sven Oliver Krumke, Luigi Laura, Maarten Lipmann, Alberto Marchetti-Spaccamela, Willem de Paepe, Diana Poensgen, and Leen Stougie, *Non-abusiveness helps: An $O(1)$-competitive algorithm for minimizing the maximum flow time in the online traveling salesman problem*, Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization, Lecture Notes in Computer Science, vol. 2462, Springer, 2002, pp. 200–214.
13. Sven Oliver Krumke, Jörg Rambau, and Luis M. Torres, *Realtime-dispatching of guided and unguided automobile service units with soft time winodws*, Proceedings of the 10th Annual European Symposium on Algorithms (Rolf H. Möhring and Rajeev Raman, eds.), Lecture Notes in Computer Science, vol. 2461, Springer, 2002, pp. 637–648.
14. C. C. Lee and D. T. Lee, *A simple on-line bin-packing algorithm*, J. ACM **32** (1985), no. 3, 562–572.
15. Daniel Dominic Sleator and Robert Endre Tarjan, *Amortized efficiency of list update and paging rules*, Communications of the ACM **28** (1985), no. 2, 202–208.
16. Shunji Tanaka, Yukihiro Uraguchi, and Mituhiko Araki, *Dynamic optimization of the operation of single-car elevator systems with destination hall call registration: Part I. Formulation and simulations*, European J. Oper. Res. **167** (2005), no. 2, 550–573.
17. ———, *Dynamic optimization of the operation of single-car elevator systems with destination hall call registration: Part II. The solution algorithm*, European J. Oper. Res. **167** (2005), no. 2, 574–587.
18. A. van Vliet, *On the asymptotic worst case behavior of harmonic fit*, J. Algorithms **20** (1996), 113–136.