Konrad-Zuse-Zentrum
für Informationstechnik Berlin

Takustraße 7
D-14195 Berlin-Dahlem
Germany

MARTIN GRÖTSCHEL, SVEN O. KRUMKE,
JÖRG RAMBAU

Online Optimization of Complex Transportation Systems

ZIB-Report 01-17 (July 2001)

ONLINE OPTIMIZATION OF COMPLEX TRANSPORTATION SYSTEMS

MARTIN GRÖTSCHEL, SVEN O. KRUMKE, AND JÖRG RAMBAU

ABSTRACT. This paper discusses online optimization of real-world transportation systems. We concentrate on transportation problems arising in production and manufacturing processes, in particular in company internal logistics. We describe basic techniques to design online optimization algorithms for such systems, but our main focus is decision support for the planner: which online algorithm is the most appropriate one in a particular setting? We show by means of several examples that traditional methods for the evaluation of online algorithms often do not suffice to judge the strengths and weaknesses of online algorithms. We present modifications of well-known evaluation techniques and some new methods, and we argue that the selection of an online algorithm to be employed in practice should be based on a sound combination of several theoretical and practical evaluation criteria, including simulation.

1. INTRODUCTION

The strategic planning of complex transportation systems such as public transportation networks, automatically guided vehicles in warehouses, etc. has received a considerable amount of attention in the last decade. *Strategic planning* is the stage of system design where an object (e.g., a telecommunication network) is designed that will remain static for a certain planning period (the network topology, and edge capacities will not change) such that a few control parameters (e.g., routing and switching) will allow an (almost) optimal handling of all input data (within a certain realistic or predicted range). The system itself is usually not yet operational when this "strategic optimization" takes place. Here, methods of *offline optimization* apply. The increasing computing power and significant advances in traditional optimization techniques have resulted in substantial savings of resources in this area.

Despite many successes of this approach, e.g., for transportation systems, it has turned out that achieving savings also requires an *optimized operational control*. Such a control involves actions to be executed while a system is running; i.e., input data arise over time, have to be processed, and (irrevocable) decisions have to be made before all input data are known. This means that methods of *online optimization* have to be employed. In many cases decision making has to satisfy certain *real-time requirements*: every decision has to be made within strict time limits.

In this paper, we survey some new methods (beyond standard competitive analysis) to obtain decision support for the choice of online algorithms in real-world transportation systems. In each case, we are looking for a "good" online control on the basis of online algorithms. The methods discussed are *competitive analysis against restricted adversaries* (a variant of competitive analysis where the offline adversary is given less power), *analysis under Δ-reasonable load* (we compare the cost of the online algorithm to a certain property of the input), *a-posteriori-analysis* (we perform an approximate, instance wise competitive analysis to compute a lower bound on the unavoidable cost), and *comparative simulation* (we compare algorithms that run simultaneously in simulation experiments).

These concepts are employed along with standard competitive analysis in real-world examples. We indicate which combination of methods could support decisions best.

Supported by the German Science Foundation, Priority Programme 469.

The rest of the paper is structured as follows: We start out by sketching our real-world examples in Section 2. In Section 3 we present the above mentioned evaluation methods for online algorithms. In Section 4 the applications of traditional and new methods to real-world systems is discussed. Section 5 summarizes what we consider the key findings of our research.

2. FOUR REAL-WORLD EXAMPLES AND WHY THEY RAISE QUESTIONS

In this section we introduce four real-world online optimization problems. One common feature is the difficulty to evaluate online algorithms for them.

The first example is the automated stacker crane in a production plant of Siemens Nixdorf Informationssysteme AG (SNI). The question is in which order the stacker crane should perform storage and retrieval operations so as to minimize the unloaded travel time. We show that for the related objective "minimize the makespan" (the time the system needs to serve a set of requests) we find a $5/2$-competitive algorithm. This is the REPLAN-heuristics already discussed in [10]. This algorithm is, however, not competitive with respect to the minimization of the total unloaded travel distance. Shall we use this algorithm anyway?

The second example studies a system of automated guided vehicles for commissioning greeting cards in a large distribution center of Herlitz PBS AG, Falkensee, one of the main distributors of office supply in Europe. Orders specifying a combination of greeting card sets have to be assigned to vehicles. These must stop at the shelf positions where the corresponding cards have to be collected ("order picking"). The question is how orders should be assigned to vehicles so that the total number of stops over all vehicles is minimized. It turns out that competitive analysis tells us nothing about which algorithm to choose in practice. For a greatly simplified problem we show that competitive analysis is even in favor of an intuitively senseless algorithm. Is there an evaluation method that proves dumb algorithms to be dumb?

The next example is a pallet elevator in the same distribution center. In this case it was already difficult to isolate a single objective function to be optimized. We decided to consider several objectives: We want to guarantee a small average and/or a small maximal flow time over a set of pallets requesting transportation. It turns out that for these objectives there is no competitive algorithm, mainly because the cost of an offline solution cannot be bounded from below. Can we evaluate algorithms without using a lower bound on the offline cost?

Finally, we investigate the integrated elevator system plus conveyor belt that distributes the pallets among the elevators. We find out that a similar analysis as in the single elevator case is still valid; the improvements, however, are leveled off by the conveyor control. Therefore, we studied an integrated optimization model for the combined elevator-conveyor system. Does this help to improve the overall performance of the system?

We will give answers to the above questions in Section 4 after we have introduced our "evaluation toolbox" in Section 3. Some of the answers are quite satisfactory, others show the need for further research.

3. MODELING AND EVALUATION TECHNIQUES

In this section, we present a sequence of methods to analyze the performance of online algorithms. The methods are ordered by decreasing mathematical strength, that is to say, the first method—if successful—yields the most rigorous analysis of the ones in this section, the last one is merely experimental. (Classical competitive analysis as described in [10] would belong to the very beginning of the section.)

3.1. **Competitive Analysis with Restricted Adversaries.** In restricting the class of algorithms for the adversary, one attempts to deal with the (justified) objection—frequently encountered against competitive analysis—concerning the unrealistic power of the adversary against which performance is measured. In standard competitive analysis the adversary is an optimal offline algorithm which has complete knowledge about the whole input in advance. There have been a number of approaches in the literature to devise "more realistic" adversary models for specific problems than the omnipotent standard offline adversary.

For the exposition we consider the online traveling salesman problem ONLINETSP on the non-negative real numbers $\mathbb{R}_0^+$ endowed with the Euclidean metric (see [10, Example 4] for the ONLINETSP in general metric spaces). The origin of the salesman is the point 0. In the ONLINETSP requests for visits to cities (points in a metric space) arrive online while the salesman is traveling. The salesman moves at unit speed and starts and ends his work at the origin 0. The objective is to find a route for the salesman which finishes as early as possible.

Each *request* is a pair $\sigma_i = (t_i, x_i)$, where $t_i \in \mathbb{R}$ is the time at which request σ_i is released (becomes known to an online algorithm), and $x_i \in \mathbb{R}_0^+$ is the point requested to be visited. It is assumed that an online algorithm does neither have information about the time when the last request is released nor about the total number of requests. An online algorithm must base its decisions at time t solely on the requests released up to time t.

Notice that the offline version of the ONLINETSP in $\mathbb{R}_0^+$ can be solved very easily even in the presence of release times (the problem is almost trivial!). However, in the online case, there does not exist an algorithm that always finds an optimal solution. More specifically, it can be shown that there is no deterministic online algorithm that achieves a competitive ratio smaller than $3/2$. The competitive ratio of $3/2$ is achieved by the following very natural and simple strategy MRIN (see [7] for the proofs):

Strategy MRIN("Move-Right-If-Necessary"): If a new request is released and the request is to the right of the current position of the server operated by MRIN, then the MRIN-server starts to move right. The server continues to move right as long as there are yet unserved requests to the right of the server. If there are no more unserved requests to the right, then the server moves towards the origin 0. □

In the lower bound construction the (standard) offline adversary abuses his power in the sense that he can move to points where he knows a request will pop up without revealing the request to the online server before reaching the point. This has motivated the concept of a "fair adversary" in the ONLINETSP: A fair adversary always keeps its server within the convex hull of the requested points released so far. As shown in [7] this adversary model indeed allows for lower competitive ratios. For instance, the above mentioned $3/2$-competitive algorithm MRIN against the conventional adversary is $4/3$-competitive against the fair adversary. In addition, one can prove the following:

theorem 3.1 ([7]). *There exists an online algorithm for the* ONLINETSP *in* $\mathbb{R}_0^+$ *with competitive ratio* $\frac{1+\sqrt{17}}{4} \approx 1.28$ *against a fair adversary. Moreover, no deterministic online algorithm can achieve a competitive ratio smaller than* $\frac{1+\sqrt{17}}{4}$ *against the fair adversary.* □

The use of a restricted adversary falls within the concept of *comparative analysis*, which was introduced by Koutsoupias and Papadimitriou [14]. The authors compare the performance of an online algorithm for the Paging Problem with that of the best paging algorithm having limited lookahead. Let Π be a minimization (online) problem. The *comparative ratio* of an algorithm ALG for Π relative to a class $\mathcal{B}$ of algorithms is defined as the worst case ratio between the solution cost produced by ALG and the best solution produced by an algorithm in $\mathcal{B}$. If $\mathcal{B}$ is the class of all offline algorithms for Π, then the comparative ratio reduces to the standard competitive ratio.

The comparative ratio has also been studied in the context of online financial problems. For most of these problems the standard adversary also appears to be too strong. To obtain meaningful (theoretical) results about the performance, e.g., of online portfolio selection algorithms, a comparison with a restricted class of offline algorithms is used. We refer to [8, Chapter 14] for details.

3.2. **Reasonable Load.** This concept was motivated by the problem of minimizing the maximal or average flow time of pallets transported by an elevator. Such a system can be modeled by the so-called *online dial-a-ride problem* ONLINEDARP. The concept of reasonable load also works in a more general setting. However, we do not want to go too much into abstraction in this paper, and we restrict our attention to ONLINEDARP, which we explain in the sequel.

We are given a metric space (X,d) with a special point $o \in X$ (the origin). Requests are triples $r = (t,a,b)$, where a is the start point of a transportation task, b its end point, and t its release time, which is—in this context—the time where r becomes known to an online algorithm. A *transportation move* is a quadruple $m = (t,x,y,R)$, where x is the starting point, y the end point, and t the starting time, while R is the set (possibly empty) of requests the server has loaded during the move. We say in this case, the move *m carries R*. The *arrival time* of a move is the sum of its starting time and $d(x,y)$. A *(closed) transportation schedule* is a sequence $(m_1,m_2,\dots)$ of transportation moves such that

- the first move starts in the origin o;
- the starting point of m_i is the end point of m_{i-1};
- the starting time of m_i carrying R is no earlier than the maximum of the arrival time of m_i and the release times of all requests in R (it maybe later, though);
- the last move ends in the origin o.

An *online algorithm* for ONLINEDARP has to move a server in X so as to fulfill all released transportation tasks without preemption (i.e., once an object has been picked up it is not allowed to be dropped at any other place than its destination), while it does not know anything about requests that come up in the future. In order to plan the work of the server, the online algorithm may maintain a preliminary (closed) transportation schedule for all known requests, according to which it moves the server. A posteriori, the moves of the server induce a complete transportation schedule that may be compared to an offline transportation schedule that is optimal with respect to some objective function (competitive analysis). For a detailed set-up see [4].

Recall that the *flow time* of a request is the difference between its completion time and its release time, while the *waiting time* is the difference between its service starting time and its release time. In the sequel, we are concerned with the following objectives:

- Minimize the *makespan* (also called the *completion time*) for the given set of requests. This is the time the server needs to fulfill all the transportation tasks.
- Minimize the *maximal flow time* (or *waiting time*) of the requests.
- Minimize the *average flow time* (or *waiting time*).

We will consider the online heuristics REPLAN and IGNORE from [10]. Since we did not choose a particular objective function yet we need to specify according to which objective function REPLAN and IGNORE will solve the corresponding offline problems. We will even evaluate REPLAN- and IGNORE-heuristics that use a different objective for the local optimization than the one that is to be minimized globally in the online problem.

Thus, for an arbitrary objective function *obj* we denote by REPLAN^{obj} resp. IGNORE^{obj} the following online heuristics:

REPLAN^{obj}: Follow the current plan. Whenever a new request becomes available compute a new plan minimizing *obj* starting at the current position.

IGNORE^{obj}: Follow the current plan; while executing it collect upcoming requests in a buffer. When done and there are non-served requests in the buffer compute a new plan for all these requests minimizing *obj*.

The motivation to consider the concept of reasonable load in this situation was two-fold.

First, competitive analysis of ONLINEDARP provides the following [4]:

- The two online heuristics $\mathsf{IGNORE}^{makespan}$ and $\mathsf{REPLAN}^{makespan}$ are $3/2$-competitive for the goal of minimizing the *makespan* of the schedule.
- For the tasks of minimizing the *maximal (or average) waiting time* or the *maximal (or average) flow time* there can be no algorithm with constant competitive ratio.
- In particular, the algorithms $\mathsf{IGNORE}^{makespan}$ and $\mathsf{REPLAN}^{makespan}$ that repeatedly minimize the *makespan* of all known requests have an unbounded competitive ratio for the overall task of minimizing the maximal or average *flow time*.

Second, in simulation studies a fundamental difference in the behavior of IGNORE and REPLAN was observed: the maximal flow times on similar inputs produced by REPLAN varied a lot while the ones produced by IGNORE were better predictable. The concept of reasonable load was developed to find a mathematical explanation of this phenomenon.

We start with some useful notation.

Definition 3.2. The *offline version* of a request $r = (t, a, b)$ is the request

$$r^{offline} := (0, a, b).$$

The *offline version* of a request set R is the request set

$$R^{offline} := \left\{ r^{offline} \ : \ r \in R \right\}. \quad \square$$

An important characteristic of a request set with respect to system load considerations is the time period in which it is released.

Definition 3.3. Let R be a finite request set for ONLINEDARP. Let the release time of a request r be denoted by $t(r)$. The *release span* $\delta(R)$ of R is defined as

$$\delta(R) := \max_{r \in R} t(r) - \min_{r \in R} t(r). \quad \square$$

Provably good algorithms exist for the makespan and the weighted sum of completion times. How can we make use of these algorithms in order to get performance guarantees for minimizing the maximum (average) waiting (flow) times? We suggest a way of characterizing request sets which we want to consider “reasonable”.

In a continuously operating system we wish to guarantee that work can be accomplished at least as fast as it is presented. In the following we propose a mathematical set-up that models this idea in a worst-case fashion. Since we are always working on finite subsets of the whole request set the request set itself may be infinite, modeling a continuously operating system.

We start by relating the release spans of finite subsets of a request set to the time we need to fulfill the requests.

Definition 3.4. Let R be a request set for the ONLINEDARP. A weakly monotone function

$$f \colon \left\{ \begin{array}{ccc} \mathbb{R} & \to & \mathbb{R}, \\ \delta & \mapsto & f(\delta); \end{array} \right.$$

is a *load bound* on R if, for any $\delta \in \mathbb{R}$ and any finite subset S of R with $\delta(S) \leq \delta$, the makespan $\mathsf{OPT}^{makespan}(S^{offline})$ of the optimum schedule for the offline version $S^{offline}$ of S is at most $f(\delta)$. In formula:

$$\mathsf{OPT}^{makespan}(S^{offline}) \leq f(\delta). \quad \square$$

Remark 3.5. If the whole request set R is finite then there is always the trivial load bound given by the makespan of R. For every load bound f, we may set $f(0)$ to be the maximum completion time we need for a single request, since nothing better can be achieved. $\square$

A stable situation would be characterized by a load bound equal to the identity on $\mathbb{R}$. In that case we would never get more work to do than we can accomplish, even if we had an optimal offline algorithm at hand.

If R has a load bound equal to a function id/ρ, where id is the identity and where $\rho \geq 0$, then ρ measures the "tolerance" of the request set: An algorithm that is by a factor ρ worse than optimal will still accomplish all the work that it gets. However, we cannot expect that the identity (or any linear function) is a load bound for ONLINEDARP because of the following observation: a request set consisting of one single request has a release span of 0 whereas in general it takes non-zero time to serve this request. In the following definition we introduce a parameter describing how far a request set is from being load-bounded by the identity.

Definition 3.6. A load bound f is called *(Δ,ρ)-reasonable* for some $\Delta, \rho \in \mathbb{R}_+$ if

$$f(\delta) \leq \frac{\delta}{\rho} \quad \text{for all } \delta \geq \Delta$$

A request set R is (Δ,ρ)*-reasonable* if it has a (Δ,ρ)-reasonable load bound. For $\rho = 1$, we say that the request set is Δ*-reasonable*, and we call a request set or a load bound *reasonable* if it is (Δ, ρ)-reasonable for some $\Delta, \rho \in \mathbb{R}_+$. □

In other words, a load bound is *(Δ,ρ)-reasonable*, if it is bounded from above by $1/\rho \cdot id(x)$ for all $x \geq \Delta$ and by the constant function with value $1/\rho\Delta$ otherwise.

Remark 3.7. If Δ is sufficiently small so that all request sets consisting of two or more requests have a release span larger than Δ then the first-come-first-serve strategy suffices to ensure that there are never more than two unserved requests in the system. Hence, the request set does not require "scheduling" the requests in order to provide for a stable system. (By "stable" we mean that the number of unserved requests in the system does not become arbitrarily large.) □

Resonable load is a plausible restriction:

Observation 3.8 (Justification of Reasonable Load). Assume, a request set for ONLINEDARP is not reasonable. Then the following holds: For all $\Delta \geq 0$ there is a request set with release span at least Δ whose offline makespan is larger than its release span.

In other words: no matter how long one collects requests there is provably no method to accomplish their service in a time equal to the collection time.

Finally, we state the theorem that mathematically shows the (somewhat surprising) fundamental difference of IGNORE^{makespan} and REPLAN^{makespan} on ONLINEDARP. (See [11] for a proof.)

theorem 3.9. *For the* ONLINEDARP *under Δ-reasonable load,* IGNORE^{makespan} *yields a maximal and an average flow time of at most* 2Δ*, whereas the maximal and the average flow time of* REPLAN^{makespan} *are unbounded.*

In Sections 4.3 and 4.4 we present practical applications where an analysis under reasonable load is possible.

3.3. **A-Posteriori-Analysis.** Competitive analysis—even in the case of existing competitiveness results—does often not provide performance guarantees that appear convincing in an efficiency oriented industrial environment. Consider a statement such as "The solution produced is in each and every situation at most 3 times worse than the optimum". Will a user be happy to hear that? Such a result is too weak in terms of the performance ratio and too strong in the sense that it covers too many (from a customer's point of view probably irrelevant) situations.

The same problem occurs in the framework of approximation algorithms: a performance guarantee for all instances of a problem is often not required. One approach that made combinatorial optimization methods have impact in real life was the delivery of *instance-wise* performance guarantees via the computation of so-called *lower bounds* for the very special instance of the problem to be solved in a particular situation.

Lower bounds can usually be derived by relaxing side constraints of a problem. The most prominent relaxation technique in combinatorial optimization is to relax the integrality constraints, thereby transforming notoriously difficult (Mixed) Integer Programs into efficiently solvable Linear Programs [10]. Optimal solutions of these may be easier to come by, and an optimal solution of the original problem cannot be cheaper than the one of the relaxed problem. On the other hand, the value of any feasible solution to the original problem yields an *upper bound* for the optimal solution. The gap between lower and upper bound at any stage of the optimization process provides, thus, an instance specific quality guarantee: the difference between the objective function values of the current feasible solution and a presently unknown optimal solution is not bigger than this gap.

In this framework the role of fast approximation algorithms is to provide for good feasible starting solutions. Good initial solutions often help to close the gap between lower and upper bounds fast and, thus, help to speed up the optimization process.

One can similarly compute lower and upper bounds for a special instance of an online optimization problem. This leads to an instance-wise competitive analysis. Since, in an online situation, a special instance is not known in advance, this kind of analysis can only be applied after all decisions have been made. Therefore, this approach is called *a-posteriori analysis*.

We now state an observation that shows what a-posteriori-analysis can achieve. We concentrate on online optimization in the time stamp model (see [10] for details). We may assume w.l.o.g. that all time stamps are positive, and we assume also that the way how a request sequence is served by an online algorithm does not influence this sequence. (This assumption is not always satisfied in real systems, since after observing how an algorithm has handled the first elements of a request sequence, the remaining requests may be altered or their order may be changed.)

Suppose that I is an instance of an online optimization problem in the time stamp model and that A is an online algorithm for this problem. Denote by $\mathsf{A}(I)$ the value of the solution A produces on I. Denote by J the corresponding instance of the offline optimization problem induced by I where all requests are known in advance and where a feasible solution has to respect all release times. Denote by K the corresponding instance of the offline optimization problem induced by I where all time stamps are removed (set to zero). Denote the optimal solution values of J and K by $\mathsf{OPT}(J)$ and $\mathsf{OPT}(K)$, respectively.

Then the following simple observation can be made.

Observation 3.10 (Justification of A-posteriori Analysis). Let I, J, K be as above. Then, under the above assumptions, there exist real numbers $c(I,\mathsf{A})$ and $c'(I,\mathsf{A})$, depending on I and on the online algorithm A, satisfying $c'(I,\mathsf{A}) \geq c(I,\mathsf{A}) \geq 1$, such that

$$\mathsf{OPT}(K) \leq \mathsf{OPT}(J) \leq \mathsf{A}(I) = c(I,\mathsf{A})\mathsf{OPT}(J) = c'(I,\mathsf{A})\mathsf{OPT}(K).$$

The above chain of equations and inequalities yields two versions of instance-wise competitive analysis depending on the chosen relaxations J or K. Usually, the quality guarantee $c(I,\mathsf{A})$ is reported as the relative gap

$$J\text{-GAP}_{\mathsf{A}} := c(I,\mathsf{A}) - 1 = \frac{\mathsf{A}(I) - \mathsf{OPT}(J)}{\mathsf{OPT}(J)}.$$

It is, however, not clear how the instance J can be solved. The corresponding combinatorial offline problems may, in fact, be hard. Even worse, it is often not apparent how to formulate these offline problems properly. The reason is that online problems in real life may come along with implicit restrictions that are difficult to model. In this sense, online problems coming from practice are sometimes “ill-posed”. In such cases, one has to relax further side constraints in addition to assuming full knowledge of the input sequence in the beginning. The resulting offline problems may then turn out to be useless in practice because of rather poor instance specific gaps. (Even for the relaxation K this is often the case.)

Thus, a-posteriori analysis is often used as follows: find relaxations between J and K that model the online restrictions as faithfully as possible and replace $\mathsf{OPT}(J)$ by the optimal objective function value of this modified problem. An example of this technique can be found in 4.1.2.

3.4. **Comparative Simulation.** The draw-back of a-posteriori analysis is that all decisions have irreversibly been made when the analysis of these decisions is available. One way out is testing the system behavior in a simulation experiment. An a-posteriori analysis can be made for every possible online algorithm. If the data used for the runs of the simulation system is "typical enough" then one can hope that a strategy whose gap in the sense of Observation 3.10 is convincingly small will behave well in reality.

Sometimes even this is too much to ask for: even in an instance-wise analysis the optimal offline algorithm may be too strong in the sense that the computed gap is quite large for every conceivable, non-clairvoyant online strategy. Then we are left with a comparison of online algorithms in simulation experiments.

One feature that makes this (somewhat "soft") method valuable is that evaluation is not limited to the computation of a single scalar objective function. Visualization of the system behavior may, e.g., help to grasp the influence of various online strategies from different perspectives: efficiency, stability, predictability, maybe others. Some of these aspects are very difficult to hard-code in a mathematical model so that the evaluation of simulation experiments by experienced human operators is still one of the most commonly accepted ways of evaluating online algorithms. We describe simulation experiments in all of our applications in Section 4.

4. The Toolbox in Action

In this section we apply the methods outlined in Section 3 including standard competitive analysis to the real-world problems sketched in Section 2. We describe the systems and the corresponding mathematical models in more detail, show that classical methods of evaluation of online algorithms are not sufficient, and apply combinations of the methods from Section 3. Where a greater level of detail is beyond the scope of this paper we provide references to the original research articles.

4.1. **Automated Stacker Cranes.** Siemens Nixdorf Informationssysteme AG (SNI) maintains a production plant where all their personal computers (PCs) and related products are assembled. Parts are brought into one of six automatic storage systems (AUSS). The AUSS serve as material buffers between the receiving area and the assembly lines located at each side of the AUSS. For each of the AUSS, there is one stacker crane fulfilling transportation tasks between the receiving buffer, the storage locations, and the buffers for the assembly line. (For a more detailed description of the layout, see [1].) The goal is to minimize the unloaded travel time of the stacker crane.

4.1.1. *Mathematical Models.* If we were to minimize the total travel time (makespan) of the stacker crane then our problem would be known as the *online stacker crane problem* ONLINESCP, a special case of the ONLINEDARP, explained in Section 3.2. Here we are concerned with a slightly different objective function.

The offline problem without release times can be modeled as an Asymmetric Traveling Salesman Problem (ATSP). An instance of ATSP consists of a complete directed graph $D = (V, A_n)$. Each node in V corresponds to a transportation task, and the the weight of the arc from v to w corresponds to the travel time from the end point of task v to the starting point of task w.

If release times have to be taken into account we are concerned with an *Asymmetric Traveling Salesman Problem* with *release times*, a special case of the Asymmetric Traveling Salesman Problem with *time windows*, the ATSPTW: here, for each request r, there is a time window $[e_r, \ell_r]$ given with a release time (*e*arliest possible start of service) e_r and a deadline (*l*atest possible completion of service) ℓ_r.

We investigated the ATSPTW because there were given deadlines for the service of requests anyway.

Time windows impose precedence constraints on the order in which the requests are served. Relaxing the time windows of an instance of the ATSPTW to the corresponding precedence constraints yields an instance of the so-called *Sequential Ordering Problem* SOP: here we are given a partial order on the set of requests and we try to find a shortest tour through all requests respecting the given partial order.

The problems ATSP, SOP, and ATSPTW are NP-hard. While much attention had already been paid to the investigation of the ATSP a thorough polyhedral study of SOP and ATSPTW was carried out for the first time in [6].

Those results were later strengthened in [1]. The goal was the design of a branch&cut algorithm able to solve on the one hand typical instances of the ATSP used for the REPLAN online heuristic and on the other hand the larger SOP resp. ATSPTW instances used for the a-posteriori analysis of several online heuristics (see Section 3.3). In the following we summarize the achievements for the SOP as an example for the polyhedral investigations contained in this article.

There are related results for the ATSPTW. We do not include them here since they are of similar nature and their statement would not shed more light on the principle situation. The interested reader may want to check again [1].

Let us start with the graph theoretic formulation of SOP. Recall that we are given a complete directed graph $D = (V = \{1,\dots,n\}, A_n)$ on n nodes with non-negative arc costs $c_{ij} \geq 0$. Moreover, in the SOP we are given a partial order "$\prec$" on V with $1 \prec i \prec n$ for all $i \in V$, w.l.o.g. A feasible solution to the SOP is a set of arcs forming a path that visits all nodes in V exactly once and that visits node i before node j whenever $i \prec j$. The goal is to find a feasible solution with minimal total arc costs.

There are several possibilities to formulate the SOP as an integer program. The polyhedral model chosen here is the following. We define the *feasible arc set* A as follows:

$$A := A_n \setminus (\{(j,i) \in A_n \; : \; i \prec j\} \cup \{(i,k) \in A_n \; : \; \exists j \in V : i \prec j \vee j \prec k\}).$$

For all feasible arcs $(i,j) \in A$ we introduce binary arc variables x_{ij} meaning that $x_{ij} = 1$ if and only if arc (i,j) is chosen to be in the solution.

With the notation

$$\begin{aligned}
x(B) &:= \sum_{(i,j)\in B} x_{ij} && \text{for } B \subseteq A,\\
A(W) &:= \left\{(w,w') \in A \; : \; w,w' \in W\right\} && \text{for } W \subseteq V,\\
\delta^+(i) &:= \{(i,j) \; : \; j \in V \setminus \{i\}\},\\
\delta^-(i) &:= \{(j,i) \; : \; j \in V \setminus \{i\}\},\\
x(i:W) &:= \{(i,w) \in A \; : \; w \in W\},\\
x(W:i) &:= \{(w,i) \in A \; : \; w \in W\}
\end{aligned}$$

an integer programming formulation of the SOP can be stated as follows:

$$\min c^T x$$

$$\text{(1)} \quad s.t. \quad x(\delta^-(i)) = 1 \qquad \forall i \in V \setminus \{1\}$$

$$\text{(2)} \quad x(\delta^+(i)) = 1 \qquad \forall i \in V \setminus \{n\}$$

$$\text{(3)} \quad x(A(W)) \leq |W| - 1 \qquad \forall W \subset V, 2 \leq |W|$$

$$\text{(4)} \quad x(j:W) + x(A(W)) + x(W:i) \leq |W| \qquad \forall i \prec j, W \subseteq V \setminus \{i,j\}, W \neq \emptyset$$

$$\text{(5)} \quad x_{ij} \in \{0,1\} \qquad \forall (i,j) \in A.$$

The object of study is the *sequential ordering polytope* SOP defined as

$$\mathrm{SOP}(n,\prec) := \mathrm{conv}\left\{x \in \mathbb{R}^A \;:\; x \text{ satisfies (1)–(5)}\right\}$$

This polytope had already been studied in [6], where new inequalities such as the predecessor/successor inequalities were derived. The following theorem summarizes the new results achieved in our project group. For details see [1].

theorem 4.1 (Offline Problems—Polyhedral Study)**.** *For* $\mathrm{SOP}(n,\prec)$ *the following hold:*

(i) If "$\prec$" is satisfies a certain regularity condition then the dimension of $\mathrm{SOP}(n,\prec)$ *equals $|A| - 2n + 3 + |F|$, where F is the set of nodes whose position in the path is fixed by "$\prec$".*

(ii) There are three types of new valid inequalities for $\mathrm{SOP}(n,\prec)$*: the strengthened D_3-inequalities, the strengthened T_k-inequalities, and the strengthened two-matching constraints.*

We refrain from explicitely listing the inequalities here because the overhead in notation would not pay off given the purpose of this article. The corresponding results on the ATSPTW can be found in [2].

4.1.2. *Evaluation of Algorithms.* The online version ONLINEATSP of the ATSP is defined in the same way as the ONLINETSP, except that the distances are not symmetric. A competitive analysis of the ONLINEATSP with the objective to minimize unloaded travel time cannot provide additional insight. The reason for this is the following: one can find request sequences that can be served by an offline algorithm without unloaded travel time and that incur a positive cost for any online algorithm. Thus, the competitiveness ratio would be infinite: not particularly helpful.

If one, however, replaces the objective "minimize total unloaded travel time" by the objective "minimize total travel time (makespan)" then—as we mentioned already—we are concerned with a special case of the ONLINEDARP. Note that these objective functions are equivalent in the sense that their function values only differ by an additive constant and that, therefore, the sets of optimal solutions are equal. From the point of view of competitive analysis, however, this change in the objective makes a huge difference.

As an application of a result in [4] we mention the following (see Theorem 4.10):

theorem 4.2 (Competitive Analysis)**.** *For the problem of minimizing the makespan of the stacker crane,* REPLAN *is* $5/2$*-competitive.*

Such a performance guarantee does not really help a decision maker. Therefore, it does make sense to evaluate the REPLAN-strategy by other means. An a-posteriori analysis was also made: we investigated the ATSP, the SOP, and the ATSPTW as relaxations in the spirit of Section 3.3.

Observation 4.3 (A-Posteriori Analysis). Real data sets from SNI provided the following a-posteriori analysis for the REPLAN-strategy repeatedly solving the ATSP of all known requests:

(i) The online solution is between 46% and 120% worse than an optimal a-posteriori solution for the ATSP.
(ii) The online solution is between 24% and 98% worse than an optimal a-posteriori solution for the SOP.
(iii) The online solution is between 3% and 72% worse than an optimal a-posteriori solution for the ATSPTW.

Since these gaps are not small enough to convince decision makers to use the REPLAN-heuristics, simulation experiments were made.

Observation 4.4 (Comparative Simulation). On real data sets the REPLAN-heuristic slightly outperforms other online heuristics such as best insertion heuristics. The at SNI previously used priority strategy with FIFO as a tie breaker performs roughly 50% worse than REPLAN; the FIFO priority algorithm is no better than a random sequencing of request.

Thus, the conclusion was to implement the REPLAN-heuristic.

4.1.3. *Implemented Solution and Practical Impact.* Although the subproblems to be solved within the REPLAN-heuristics are NP-hard, there are codes available so that the REPLAN-heuristics can be used in real-time situations in practice. In order to obtain an any-time algorithm [10], we implemented an optimization process working in three phases:

Phase 1:: Perform cheapest insertion (BESTFIT).
Phase 2:: Run a random insertion. Then pick the winner of Phase 1 and 2.
Phase 3:: Solve the ATSP to optimality (branch&bound from [2]) and replace the old sequence completely by the optimal one (REPLAN).

Phase 1 runs in time linear in the number of requests and is always completed. For the typical problem sizes that occur in our application (the number of requests is less than 60) this is done in fractions of a second. Even Phase 3 could always be completed within a few seconds. If the stacker crane becomes idle before Phase 3 is finished, the optimization process is interrupted, and the best sequence found so far is passed to the stacker crane.

Our simulation experience showed that REPLAN empirically gives the best results on average. SNI provided data for one week of production. During this period on one AUSS each generated task and each move of the stacker crane were recorded. It turned out that in heavy load periods the times needed for unloaded moves could be reduced by approx. 30%.

As a result the optimization package was put in use on five AUSS, and the results were confirmed in everyday production.

4.2. **Commissioning of Greeting Cards.** One of the commissioning departments in the distribution center of Herlitz is devoted to greeting cards. The cards are stored in four parallel shelving systems. In accordance with the customers' orders, the different greeting cards are collected in boxes that are eventually shipped to the recipient. Order pickers on eight highly automated guided vehicles collect the orders from the storage system, following a circular course. The vehicles are unable to pass each other. Moreover, due to security reasons, only two vehicles are allowed to be in the middle aisles at the same time, whereas three are allowed in the first and last aisle.

At the loading zone, each vehicle is logically "loaded" with up to 19 orders from a pool that changes over time. A dispatcher decides when to send a vehicle onto the course. After leaving this area the vehicles automatically stop at a position where cards have to be picked from the shelf according to the logical load. The goal is the minimization of the makespan of all requests generated on one day subject to some side constraints explained in [3, 13]. Congestion among the vehicles should be avoided. This is important because congestions lead to undesirable side-effects (that are very difficult to evaluate mathematically). These include human order pickers leaving for an extra-break when their vehicles run into congestions. (For more details consult [13].)

4.2.1. *Mathematical Models.* For the theoretical analysis it is necessary to provide a proper mathematical formulation of the problem under consideration. We remark again that the modeling phase may already result in a heuristic approach because the practical problem comes in day-to-day terms that have no straight-forward mathematical translations. The Commissioning Vehicle Routing Problem (CVRP) to be considered in the competitive analysis in Section 4.2.2 is the following.

An instance of CVRP consists of a set $L = \{1, 2, \ldots, m\}$, the pick positions, and a set of empty vehicles $v_1, \ldots, v_q$, each with capacity C. A request sequence $\sigma = r_1, r_2, \ldots$ consists of a chronologically ordered collection of sets of pick positions.

A vehicle to which C requests have been assigned is replaced by a new empty vehicle. In the online situation we require that request r_i is permanently assigned to vehicle $v(r_i)$ before r_{i+1} becomes known and that the length of the sequence is unknown until the last request comes in. That means, CVRP is an online problem in the sequence model (see [10] for basic facts on online problems).

For a sequence of requests, a solution to the CVRP is an assignment of every request r_i to a vehicle $v(r_i)$ so that the number of requests assigned to each vehicle does not exceed C. The objective is to minimize the total number of pick positions assigned to the vehicles. In [13] it was shown that the offline version of CVRP with no release times is already an NP-hard problem. In fact, solving the corresponding integer program in reasonable time turned out to be out of reach for commercial software packages like CPLEX.

One explanation for the intrinsic difficulty of this variation of a capacitated assignment problem is given by the following result that we state informally here (see [13] for details):

theorem 4.5 (Offline Problem)**.** *The optimal solution of a (certain) linear-programming relaxation of* CVRP *corresponds to the evenly distributed fractional assignment, i.e., every request is partially assigned to each available vehicle.*

This observation yields that the linear programming relaxation does not provide any exploitable information on how to assign requests to vehicles.

4.2.2. *Evaluation of Algorithms.* In [13] the following result was shown:

theorem 4.6 (Competitive Analysis)**.** *The following hold for the* CVRP*:*

(i) Any rule for the assignment of requests to vehicles yields a C-competitive algorithm for the CVRP*, where C is the capacity of a vehicle.*
(ii) No online algorithm for the CVRP *can be better than* 2*-competitive.*
(iii) The algorithm BESTFIT*—set $v(r_i)$ to the vehicle whose number of pick positions gets the least increase—is no better than C-competitive.*

In other words: competitive analysis does not provide much insight. In particular, the intuitively "reasonable" BESTFIT-heuristic is, from a competitive analysis point of view, not better than any stupid rule.

Even worse: recent investigations showed that even for a substantially simplified version of the CVRP we run into the odds of competitive analysis. In the following excursion into theoretical online optimization we sketch the result.

Consider the following *Online Bin Coloring Problem* ONLINEBC: We are given a natural number $q > 0$, infinitely many numbered bins with volume capacity C, and a sequence of requests $r_1, r_2, \ldots$ consisting of colored items of unit volume. We have to place the items into the bins so that, at any time, no more than q bins contain more than zero and less than C items. We have to stuff r_i into a bin before we get to know r_{i+1} (sequence model). The goal is to minimize the number of colors in the most colorful bin, i.e., the maximum number of items of distinct colors in a bin over all bins.

This translates to the language of commissioning as follows: every request has only one stop position, and we try to minimize over all vehicles the maximal number of stops of a vehicle, rather than the total number of stops. (This is a useful objective that helps to balance the vehicle load and, thus, to reduce congestion).

Consider the following online algorithms for ONLINEBC:

- Algorithm ONEBIN puts all items into one single bin until it is full. Then it picks another bin etc. (This is a truly dumb algorithm.)
- Algorithm BESTFIT puts every item into the bin that already contains that color, if such a bin exists. Otherwise, it puts the item into the bin with the least number of colors so far, with ties broken arbitrarily.

The following theorem shows that standard competitive analysis is problematic for this class of problems:

theorem 4.7 (Competitive Analysis)**.** *The following hold for the* ONLINEBC*:*

(i) BESTFIT *is* $\min\{C, 2q + \lfloor (qC - 3q + 1)/C \rfloor\}$*-competitive.*
(ii) ONEBIN *is* $\min\{C, (2q-1)\}$*-competitive.*
(iii) BESTFIT *is no better than* $2q$*-competitive whenever* $C \geq 2q^3 - q^2 - q + 1$.
(iv) No deterministic online algorithm can be better than $O(q)$*-competitive.*

This proves that competitive analysis does not provide any hint as to which algorithm should be chosen in practice, even in the restricted models of this section.

4.2.3. *Implemented Solution and Practical Impact.* Several heuristics that reduce the total number of stops and distribute them evenly among the vehicles were implemented. These are versions of the BESTFIT algorithm, together with local exchange heuristics. The computation times of these algorithms are short so that they can be run in a real-time situation.

We implemented a detailed simulation model for the whole commissioning area in which we compared our approach to the one used so far. Herlitz provided production data from a period of about six weeks, which were the basis for the comparison. The main results are the following:

- A significant improvement with respect to the completion times of the orders can be achieved.
- The number of vehicles, used at Herlitz, can be reduced from eight to six without any negative impact on the system performance.
- Congestions over a few seconds can be avoided completely.

We conclude that BESTFIT—although not distinguished in the competitive analysis—was the basis for significant improvements in practice. The simulation results convinced Herlitz to test a prototype of the simulation program as a decision support tool for the dispatcher.

4.3. **Elevators.** The automated pallet transportation system in the Europe-wide distribution center of Herlitz PBS AG has been designed to handle all pallet transportation taks from/to the receiving docks, the production and commissioning departments, the automated shelf system, and the loading dock from where the products are shipped to the customers by trucks. This pallet transportation network runs on nine floors and is quite complex. The overall goal is to run the operations "smoothly", a mathematically not well-defined term that means something like: each individual transportation task should be executed quickly, time windows (existing for some of the tasks) should be observed, and the whole system should be congestion free. The last objective may be in conflict with the others, and a difficulty is to find an appropriate balance.

We address here the elevators, one of the building blocks of the pallet transportation system. There are two systems of five elevators. Each elevator can carry at most one pallet. Transportation requests occur (unpredictably) throughout the day and are somehow distributed to the elevators. Congestion does frequently occur at the entry points and should be avoided by running the elevators "well". Of course, congestion depends on both the assignment of requests to the elevators and on the control of the elevators. We discuss here the second issue.

At Herlitz, each elevator is controlled independently from the others; there is no "master control" watching over the whole elevator system simultaneously. It is therefore clear that optimizing the individual elevators may not result in the desired congestion-free system, but it will at least help running the system faster. We decided to investigate the following problems for individual elevators and systems of elevators (compare to Section 3.2):

- Minimize the *makespan* for the given set of requests.
- Minimize the *maximal flow time* of the requests.
- Minimize the *average flow time* of the requests.

While the makespan is a measure for how fast the system is as a whole the other two objectives are rather a measure for the speed of the system as "experienced" by the individual pallets. Note that in contrast to the makespan the maximal and average flow times also make sense in a continuously operating system, i.e., with infinite request sets.

4.3.1. *Mathematical Models.* The basic model chosen for investigating algorithms for the control of elevators is the ONLINEDARP, which we introduced in Section 3.2. In the sequel we first investigate the control of a single elevator. Briefly, this is the problem of how to serve online transportation requests in a metric space which is a path, where the server is assumed to have capacity one.

In the context of pallet transportation there is a subtle additional side constraint involved: we do not have random access to the pallets waiting on a particular floor. That means that requests from the same floor need to be scheduled in their order of appearance, while requests on different floors can still be shuffled. This leads to the problem ONLINE-FIFODARP. Here the subset of requests occuring at a particular point in the metric space must be served in the order of appearance.

As an extension of ONLINEDARP we also investigated the corresponding problem with capacity larger than one, the ONLINECDARP.

In order to be able to use REPLAN- or IGNORE-heuristics for any of the online problems in real-time we need to find efficient algorithms for the corresponding offline problems. In the following theorem we summarize the results:

theorem 4.8 (Offline Problems)**.** *The following complexity results hold for the dial-a-ride problems under consideration:*

(i) There is a polynomial time algorithm for DARP *on paths.*
(ii) DARP *on trees (even on so-called caterpillars) is NP-hard.*
(iii) There is a polynomial time algorithm for FIFODARP *on paths.*
(iv) CDARP *is NP-hard on paths.* □

theorem 4.9 (Offline Problems)**.** *The following approximation results hold:*

(1) There is a $5/3$*-approximation algorithm for* FIFODARP *on trees.*
(2) There is a $9/4$*-approximation algorithm for* FIFODARP *on general graphs.*
(3) There is a 3*-approximation algorithm for* CDARP *on paths.* □

The observed performances of the approximation algorithms for FIFODARP on instances occuring in the online situation (e.g., while applying the REPLAN-heuristics) are much better. Therefore, these approximation algorithms can be used to produce a starting solution for a branch&bound procedure to find reasonably good offline solutions to feed the REPLAN-heuristics in real-time.

4.3.2. *Evaluation of Algorithms.* Motivated by results on the ONLINETSP in [5] we carried out a competitive analysis for ONLINEDARP for the minimization of the makespan. The results are the following:

theorem 4.10 (Competitive Analysis)**.** *For the problem of minimizing the makespan in* ONLINEDARP *the following hold:*

(i) No deterministic online algorithm can be better than 2*-competitive. (This follows easily from* [5].*)*
(ii) The REPLAN- *and the* IGNORE*-heuristic are* $5/2$*-competitive.*
(iii) There is a 2*-competitive algorithm (called* SMARTSTART *in* [4]*).*

In other words, we found one optimally competitive online algorithm for our problem.

For the other objective functions, the approach via competitive analysis yields the strongest conceivable negative result, i.e., no decision support at all:

Observation 4.11 (Competitive Analysis). There are no competitive algorithms for the tasks of minimizing the maximal or average flow times in ONLINEDARP.

The concept of reasonable load (see 3.2) was developed to get at least a weaker performance evaluation. We have already seen two canonical online heuristics in that section: REPLAN^{makespan} and IGNORE^{makespan}. Recall that both work by repeatedly minimizing the makespan: while REPLAN^{makespan} computes a new plan whenever a new request becomes available, IGNORE^{makespan} does not compute a new plan before the old plan is completely served. What about REPLAN^{maxflow}, REPLAN^{avgflow}, IGNORE^{maxflow}, IGNORE^{avgflow}? What about the problems ONLINECDARP and ONLINEMDARP (more than one server)? Some answers are collected in the following theorem:

theorem 4.12 (Analysis Under Reasonable Load)**.** *For* ONLINEDARP*,* ONLINEFIFO-DARP*,* ONLINECDARP*,* ONLINEFIFOCDARP*,* ONLINEMDARP*,* ONLINEFIFOMDARP *the following hold under* Δ*-reasonable load:*

(i) The maximal and average flow times of IGNORE^{makespan} *are at most* 2Δ*.*
(ii) The maximal and average flow times of REPLAN^{makespan} *maybe arbitrarily large.*
(iii) The maximal and average flow times of REPLAN^{avgflow} *maybe arbitrarily large.*

We do not know the performance of REPLAN^{maxflow} at present. We have, however, found another provably good algorithm that imposes additional restrictions on the repeatedly computed plans. We assume that this algorithm, called DELTAREPLAN, knows Δ. The algorithm DELTAREPLAN follows the current plan. Whenever a new request comes up DELTAREPLAN computes a new plan minimizing the makespan subject to the condition that all requests in the plan have a flow time of no more than 2Δ. If the optimal plan is shorter than Δ then it is accepted as the new plan. Otherwise it is rejected, and the algorithm proceeds with the old plan. When the old plan is done, a new plan is accepted in any case.

We could prove the following in [9]:

theorem 4.13 (Analysis Under Reasonable Load)**.** *For* ONLINEDARP*,* ONLINEFIFO-DARP*,* ONLINECDARP*,* ONLINEFIFOCDARP*,* ONLINEMDARP*,* ONLINEFIFOMDARP *the following holds under* Δ*-reasonable load:*

The maximal and average flow times of DELTAREPLAN *are at most* 2Δ*.*

This theorem motivates the problem of finding out the Δ while working on a Δ-reasonable request set. Observe that for, e.g., IGNORE^{makespan} it is not necessary to have information on the correct Δ.

Assume that DELTAREPLAN dynamically computes and uses an approximation $\tilde{\Delta}$ of Δ while working on a Δ-reasonable request set. If always $\tilde{\Delta} = 0$ then we observe that all plans are rejected and the algorithm behaves like IGNORE^{makespan}, thus the performance guarantee in Theorem 4.12 takes effect. More general: whenever we underestimate Δ then DELTAREPLAN achieves the same performance guarantee as in Theorem 4.13.

In the following we define a modification DYNDELTAREPLAN of DELTAREPLAN that needs not know the real Δ. Algorithm DYNDELTAREPLAN works similar to DELTAREPLAN except that it computes a dynamically changing $\tilde{\Delta}$. This $\tilde{\Delta}$ is defined to be the makespan of the latest accepted plan. The first value for $\tilde{\Delta}$ is the length of the first plan computed. Whenever a new request occurs DYNDELTAREPLAN computes a potential new plan with all flow times at most $2\tilde{\Delta}$. If the makespan of the potential plan is at most $\tilde{\Delta}$ then DYNDELTAREPLAN accepts it as the new plan.

The following result could be achieved.

theorem 4.14 (Analysis Under Reasonable Load)**.** *For* ONLINEDARP, ONLINEFIFODARP, ONLINECDARP, ONLINEFIFOCDARP, ONLINEMDARP, ONLINEFIFOMDARP *the following holds under* Δ*-reasonable load:*

The maximal and average flow times of DYNDELTAREPLAN *are at most* 2Δ.

A "heuristic reason" for the correctness of this result is the following: whenever we underestimate Δ we may get fewer accepted new plans. But whenever no new plan is accepted and the old plan is accomplished we are working like IGNORE^{makespan}, which is fine because of Theorem 4.12.

In order to get some idea how the investigated algorithms behave on the average with respect to speed, stability, and predictability we carried out simulation studies for the basic elevator control problem. In addition to our algorithms we tested the heuristics FIFO and NN. The latter one always serves the nearest request next. Moreover, we included the heuristic NN-MAXAGE. This heuristic works like NN except that whenever a request is older than a maximal age parameter this request has to be served next. These three heuristics are implemented as possible elevator controls in the Herlitz system.

Observation 4.15 (Comparative Simulation). A simulation experiment on several random data sets for the ONLINEDARP yielded the following results:

- The FIFO-heuristic is suitable only for very low load situations. Otherwise, the maximal and the average flow times explode; heavy system congestion is apparent.
- The NN-heuristic produces very low average flow times on the average. The maximal flow times are—especially in medium load situations—unpredictable, i.e., sometimes very high.
- The NN-MAXAGE-heuristic cures the problem of unreliability of NN only in low load situations. In high load situations it suddenly behaves like the FIFO-heuristic and leads to heavy system-congestion.
- The REPLAN^{makespan}-heuristic shows mostly good average flow times. Its maximal flow times are comparable to NN, i.e., at times very bad.
- The IGNORE^{makespan}-heuristic produces slightly worse average flow times than NN or REPLAN^{makespan}. The maximal flow times, however, are among the best for all load situations. This heuristic is in a sense self-calibrating.
- The DYNDELTAREPLAN-heuristic behaves like IGNORE^{makespan} but shows on the average a little bit worse maximal flow times and slightly better average flow times.

The additional benefit of the simulation studies over a mere evaluation of an objective function is the possibility of watching the system behavior as a whole. The algorithm that is chosen eventually depends on the preferences of the administrator of the system under consideration. At Herlitz, there is a strong focus on stability over mere speed so that IGNORE^{makespan} and related heuristics seem suitable.

4.4. **Integrated Elevator Systems.** We mentioned in the previous section that the software at the Herlitz plant does not support a so-called synchronized pallet transportation. This means the controls for the individual elevators make their decisions without taking into account each other's and the conveyor system's states. Thus, we investigated the control of single elevators as discussed in the previous section. The interplay between these modules of the transportation system is not negligible, though.

In simulation studies where the conveyor belts from and to the elevators were included in the simulation system we found out that many effects observed for single elevators are leveled out. This motivated the investigation of the integrated system of conveyor belts and multiple elevators. Since the software base of the transportation system at Herlitz cannot be changed easily; research results in this area do not have direct bearing in practice.

Having this in mind we simplified the layout of the combined conveyor/elevator system in order to approach an integrated system control in reasonable steps. At Herlitz, on each floor, the conveyor system lets the pallets move on a circular belt with one entry and one exit to the production and commissioning area. There are five elevators in the interior of the circle. The pallets can reach and leave the corresponding waiting slots via switches. The waiting/leaving slots have capacity one. A pallet may move to the waiting slot only if the corresponding leaving slot on its destination floor is empty.

The coupling in this system is very difficult to model. Moreover, one question that arises in this context is whether layouts of this type are suitable for efficient control. Thus, we started our investigation on the basis of the following hypothetical layout: Pallets line-up in a waiting queue of infinite capacity. Behind that queue they enter separate waiting queues in front of the elevators. We call this problem *online multi server sequential ordering problem*, ONLINE m-COST-SOP for short. The task is to distribute pallets online to the elevator queues and to control the elevators so that the maximal or average flow times are minimal.

The idea is to use a variant of the IGNORE^{makespan}-heuristic. This requires minimizing the makespan in the corresponding offline problem. In this case, the makespan is the time when the last elevator has finished. In contrast to the case of single elevators, not all types of REPLAN-heuristics can be employed (at least not in a straight-forward form) because of the following problem: Once a set of pallets is distributed among the elevator queues the pallets will immediately move into their queues. Because the pallets cannot change the elevator the decision which elevator a particular pallet should take can not be revised.

4.4.1. *Mathematical Models.* The main idea is to model the problem as an ATSP on the request digraph (cf. Section 4.1.1) with two modifications: first, there is more than one server. Second, the pallets in the waiting queue at a particular elevator on some floor need to be served in a FIFO order. Each of these generalizations of the ATSP has been studied already in the literature: the first one in the case of a single server type was reduced to the single server case in [15]; in a more general form for servers with distinct properties (m-COST-ATSP) it was studied in [12]. The second one was already discussed in Section 4.1.1. We decided to investigate the combined problem m-COST-SOP: the *multi server sequential ordering problem*.

There is one further subtlety involved: since in the m-COST-SOP-model the maximal length of a tour in the request graph over all servers is minimized we need to take into account the loaded travel time in the arc costs. Otherwise we might get a solution where all the servers have similar unloaded travel times but their total travel times (makespans) may vary a lot and the makespan of the whole system is not optimal at all. That means: in the case of more than one server minimizing the makespan and minimizing the unloaded travel times are no longer equivalent.

Having this in mind, our model is almost the same as the m-COST-ATSP in [12] except that it also contains the corresponding precedence forcing constraints. These look like the constraints (4) in 4.1.1. We do not want to reproduce the complete model here. We just state that several properties of the SOP and the m-COST-ATSP survive in their common generalization m-COST-SOP.

theorem 4.16 (Offline Problems—Polyhedral Study)**.** *The following hold for the m-COST-SOP-polytope:*

- *The dimension of the* m-COST-SOP*-polytope for regular precedences equals* $m(n^2 - |R|) - n$*, where* R *is the set of comparable pairs of nodes.*
- *Modified versions of the so-called* σ, π, v_0-σ *and* v_0-π*-inequalities are valid for the* m-COST-SOP*-polytope.*
- *Facets of the one-server subproblem of* m-COST-SOP *can be lifted to facets of the* m-COST-SOP.

Computational experiments have shown that the integrated optimization of all servers yields an improvement in the unloaded travel times of 50% on the average.

4.4.2. *Evaluation of Algorithms.* It turns out that, also for the ONLINE m-COST-SOP, the analysis under reasonable load is analogous to the previously discussed cases.

theorem 4.17 (Analysis Under Reasonable Load)**.** *The maximal and average flow times of* IGNORE^{makespan} *for the* ONLINE m-COST-SOP *under* Δ*-reasonable load are at most* 2Δ*.*

This theoretical result is hard to implement in a real-time compliant way: the m-COST-SOP turned out to be very difficult. It rarely happens that one can find optimal solutions for instances with 20 requests in less than a minute. The real-time restrictions on an elevator control rather require answers within seconds. Thus, only heuristic solutions can be used in the online situation. Evaluation of such heuristics is research in progress.

There is one other strong argument against an unmodified use of IGNORE^{makespan}: all servers but one would very frequently wait idle for the last server to finish its part of the plan. This can be by-passed by, e.g., letting the servers work on some requests inbetween. Still, the theoretical analysis matches reality much less than in the single server case.

Preliminary simulation studies on the basis of simple heuristics for the m-COST-SOP and on modified IGNORE- and NN-heuristics are no longer in favor for the IGNORE-approach for certain parameter settings.

This shows among other things that it is quite hard to find a well-performing online control of an integrated transportation system.

5. CONCLUSION

We have discussed new evaluation methods for online optimization problems on the basis of four real-world examples. I turns out that, usually, only a combination of such methods is able to deliver convincing advice to decision makers.

To meet real-time requirements fast offline optimization algorithms are needed, in general, as building blocks for the online heuristics such as IGNORE and REPLAN. We have, e.g., introduced fast approximation algorithms for DARP that enable us to run these heuristics in real-time in the elevator control problem.

We have shown that, for the evaluation of online algorithms, classical competitive analysis may lead to either void conclusions (all algorithms are equally bad for the minimization of flow times for ONLINEDARP) or may even be in favor of a senseless algorithm (ONEBIN is best possible for ONLINEBC). New methods such as analysis under reasonable load provide new insight in some of these cases. For example, we could tell which of the two online heuristics IGNORE and REPLAN is more suitable with respect to the minimization of flow times for the ONLINEDARP.

The observation of the system behavior as a whole in simulation experiments is still unavoidable because, this way, it is possible to monitor more complex effects than the projection to a one-dimensional objective function can possibly detect.

REFERENCES

1. N. Ascheuer, *Hamiltonian path problems in the on-line optimization of flexible manufacturing systems*, Ph.D. thesis, Technische Universität Berlin, 1995.
2. N. Ascheuer, M. Fischetti, and M. Grötschel, *Solving the asymmetric travelling salesman problem with time windows by branch-and-cut*, Preprint SC 99-31, Konrad-Zuse-Zentrum für Informationstechnik Berlin, 1999.

3. N. Ascheuer, M. Grötschel, S. O. Krumke, and J. Rambau, *Combinatorial online optimization*, Proceedings of the International Conference of Operations Research (OR'98), Springer, 1998, pp. 21–37.
4. N. Ascheuer, S. O. Krumke, and J. Rambau, *Online dial-a-ride problems: Minimizing the completion time*, Proceedings of the 17th International Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Computer Science, vol. 1770, Springer, 2000, pp. 639–650.
5. G. Ausiello, E. Feuerstein, S. Leonardi, L. Stougie, and M. Talamo, *Algorithms for the on-line traveling salesman*, Algorithmica (2001), To appear.
6. E. Balas, M. Fischetti, and W. Pulleyblank, *The precendence constrained asymmetric traveling salesman polytope*, Technical Report 15213, Carnegie Mellon University, Pittsburgh, 1992.
7. M. Blom, S. O. Krumke, W. E. de Paepe, and L. Stougie, *The online-TSP against fair adversaries*, Proceedings of the 4th Italian Conference on Algorithms and Complexity, Lecture Notes in Computer Science, vol. 1767, Springer, 2000, pp. 137–149.
8. A. Borodin and R. El-Yaniv, *Online computation and competitive analysis*, Cambridge University Press, 1998.
9. B. Glück, *Online-Steuerungen automatischer Transportsysteme bei vertretbarer Belastung*, Diplomarbeit, Technische Universität Berlin, 2000.
10. M. Grötschel, S. O. Krumke, and J. Rambau, *Forschungsartikel*, ch. This book, Springer, 2001.
11. D. Hauptmeier, S. O. Krumke, and J. Rambau, *The online dial-a-ride problem under reasonable load*, Proceedings of the 4th Italian Conference on Algorithms and Complexity, Lecture Notes in Computer Science, vol. 1767, Springer, 2000, pp. 125–136.
12. C. Helmberg, *The m-cost ATSP*, Proceedings of the 7th Mathematical Programming Society Conference on Integer Programming and Combinatorial Optimization, Lecture Notes in Computer Science, vol. 1610, Springer, 1999, pp. 242–258.
13. N. Kamin, *On-line optimization of order picking in an automated warehouse*, Ph.D. thesis, Technische Universität Berlin, 1998.
14. E. Koutsoupias and C. Papadimitriou, *Beyond competitive analysis*, Proceedings of the 35th Annual IEEE Symposium on the Foundations of Computer Science, 1994, pp. 394–400.
15. G. Reinelt, *The traveling salesman – computational solutions for tsp applications*, Lecture Notes in Computer Science, vol. 840, Springer, 1994.

MARTIN GRÖTSCHEL, KONRAD-ZUSE-ZENTRUM FÜR INFORMATIONSTECHNIK BERLIN

SVEN O. KRUMKE, KONRAD-ZUSE-ZENTRUM FÜR INFORMATIONSTECHNIK BERLIN

JÖRG RAMBAU, KONRAD-ZUSE-ZENTRUM FÜR INFORMATIONSTECHNIK BERLIN