Stability with uniform bounds for online dial-a-ride problems under reasonable load

Sven Oliver Krumke and Jörg Rambau

Abstract In continuously running logistic systems (like in-house pallet transportation systems), finite buffer capacities usually require controls achieving uniformly bounded waiting queues (strong stability). Standard stochastic traffic assumptions (arrival rates below service rates) can, in general, not guarantee these strong stability requirements, no matter which control. Therefore, the worst-case traffic notion of reasonable load was introduced, originally for the analysis of the Online-Dial-a-Ride Problem. A set of requests is reasonable if the requests that are presented in a sufficiently large time period can be served in a time period of at most the same length. The rationale behind this concept is that the occurrence of non-reasonable request sets renders the system overloaded, and capacity should be extended. For reasonable load, there are control policies that can guarantee uniformly bounded flow times, leading to strong stability in many cases. Control policies based on naive reoptimization, however, can in general achieve neither bounded flow times nor strong stability. In this chapter, we review the concept and examples for reasonable load. Moreover, we present new control policies achieving strong stability as well as new elementary examples of request sets where naive reoptimization fails.

1 Introduction

Consider a distribution center with various floors, connected to a warehouse for standard pallets. Pallets are moving horizontally along conveyor belts and vertically in elevators. (One such system can be found in a distribution center of Herlitz for office supply in Falkensee near Berlin.) By design of the micro-control, a pallet can only move forward on a conveyor belt, if there is enough space in front of it. A

Sven Oliver Krumke
Technische Universität Kaiserslautern, e-mail: krumke@mathematik.uni-kl.de

Jörg Rambau
Universität Bayreuth, e-mail: joerg.rambau@uni-bayreuth.de

section of a conveyor belt completely filled with pallets results in a complete standstill of that belt. In such an event, pallets have to be removed manually in order to resume transportation.

For example, if pallets requesting an elevator stay too long in the subsystem available for waiting pallets, that subsystem will seize to work. Thus, the goal is to control an elevator in such a way that the waiting slot capacities are never exceeded. The number of pallets in such waiting slots can be minimized by minimizing the average flow times (also called sojourn times) of the pallets waiting for an elevator. Another requirement is that the flow time of an individual pallet is not arbitrarily large (infinite deferment), since such a forgotten pallet can hold back the delivery of a large order. Figure 1 shows the single waiting slots in front of some pallet elevators in the Herlitz distribution center.

Fig. 1 Pallet elevators at the distribution center of Herlitz with a single waiting slot in front of each

There are several mathematical theories that suggest a framework for the analysis of this system. Queuing theory [10] captures particularly well average entities in the steady state of such a system. It utilizes stochastic information on the input stream of requests. However, it is extremely difficult to derive non-trivial control policies from it, since usually policies are an input and not an output of the computations. Stochastic dynamic programming in Markov Decision Processes [19] is in principle suitable for finding an control policy for stochastic inputs that is optimal in expectation, but in this case the curse of dimensionality renders a direct application impossible. For example, an elevator system with one elevator of capacity one and with n floors containing w waiting slots each requires a state space of at least n^{nw+1}

states. For one waiting slot on each of eight floors—as can be found at Herlitz—this means more than 134 million states, and this does not yet include state information on already accumulated waiting times in the system. Approximate dynamic programming [5, 6] yields policies that are reported to perform well in experiments, but performance guarantees cannot be given.

An interest in worst-case results rather than expected performance measures then triggered the concept of *competitive analysis* [4]: the worst case results computed by min-max dynamic programming are often meaningless because in the worst-case all policies are equally and maximally bad. See, for example the famous paging problem [4, 7], where, obviously, each paging policy may have page fault at every page request. To present a principle solution to this dilemma, the more game-theoretic competitive analysis was developed [4, 7] and founded the area of online-optimization in algorithmic theory. In online-optimization, an *online algorithm* is presented a sequence of requests. The online algorithm has to answer the requests in such a way that an answer at any point in time is a function of all requests and answers given up to that time, i.e., the online algorithm has no clairvoyant abilities. In competitive analysis, the cost of an online algorithm on a *particular request sequence* is compared to the cost of an *optimal offline algorithm* on *the same request sequence*. In contrast to the online algorithm, the optimal offline algorithm is clairvoyant, i.e., it knows the whole sequence in advance and computes an optimal sequence of answers. Both have unlimited computing power in order to stress the fact that we want to evaluate how well an online algorithm handles the lack of information about future requests. The supremum of the cost ratios over all request sequences is the *competitiveness* of the online algorithm. The infimum of the competitivenesses over all online algorithms is the competitiveness of the online problem.

However, for many practical problems, the so-called triviality barrier is met: all online algorithms are equally bad if compared to an optimal offline algorithm. This is, unfortunately, also the case in our application: the competitiveness of our problem is infinite. The reason for this is simple: there are arbitrarily long request sequences where the optimal offline algorithm can always be exactly at the floor where a new request arrives (i.e., no waiting time), whereas this is impossible for any online algorithm (i.e., positive waiting time). Thus, the cost ratio depends on the request sequence, i.e., there is no uniform bound.

Recent progress has been made using *stochastic dominance* as a means for performance comparison of online algorithms [14]. Stochastic dominance is a very strong statement. Thus, the problems that could be tackled by this so far are quite elementary.

Despite these difficulties in the theoretical analysis, experience shows a satisfactory performance of so-called *replan algorithms* in online optimization. These are online algorithms that resemble the paradigm of a receding-horizon in Model Predictive Control (MPC): in a closed-loop, use a control that is computed as the optimum open-loop control over some finite horizon. The development of the system is estimated on the basis of a model of the system behaviour and assumptions on the possible future inputs and/or disturbances. Since future requests in online op-

timization are completely unknown and no deterministic prediction is likely enough to become true, replan algorithms in online optimization usually perform their computations with no future requests at all. The model prediction then restricts itself to the forecast of future events when the schedule is carried out with no new requests arriving. We assume in this chapter that this part of the prediction is exact.

In our application the structure of a generic replan algorithm is informally described as follows. At a particular point in time, we consider only pallets known to the system (a *system-snapshot*). We then compute an open-loop scheduling that is optimal with respect to carefully engineered constraints and a carefully engineered objective function (the *snapshot problem*). This scheduling is "used" until a new request enters the system. The new request then triggers a new optimization computation. What does it mean to "use" a schedule? In order to interpret a policy utilizing pre-computed schedules as a control policy for the elementary elevator movements, the schedules need to be translated in sequences of microscopic controls like "Go Up/Down one floor", "Halt at current floor", "Let pallet enter/exit". Whereas in the online optimization community this transformation of schedules and routings into low-level controls is rarely mentioned explicitly, there are more detailed descriptions of this process in the MPC literature [22]. We call the resulting control policy a *replan policy*. Since we are assuming that carrying out a schedule happens with no disturbances, this transition is straight-forward and is not made explicit in the following. Therefore, the terms *policy* and *online algorithm* can be understood interchangeably in the following.

The arguably most straight-forward replan policy is what we call *naive reoptimization*. It assumes that there is a so-called *associated offline-version* of the problem. This is an offline-optimization problem that describes the best possible operation of the system given all inputs throughout an evaluation period. Such an evaluation period may be one day for the elevator system in our application. The offline-version should have the property that an optimal solution to it corresponds to a best possible control of the system when the inputs are exactly as predicted. The (non-implementable) policy given by a solution to the offline-version with exactly the actual future requests can be interpreted as a policy used by a clairvoyant controller that optimizes controls under exact predictions of the future on the whole evaluation period. *Naive reoptimization* is the (implementable) policy that uses the same constraints and objective as in the offline-version; the inputs, however, are restricted to all currently known inputs. Such a policy is optimal when no further inputs occur. In our case: when no additional requests arrive. Note, that on the controller level this policy still uses non-trivial predictions because carrying out a schedule through all known requests usually takes many stages of low-level controls.

In order to utilize sensible stochastic forecasts about incoming requests, all replan policies can in principle be enhanced by incorporating a probability distribution over a limited number of future requests. This, however, leads to extremely hard-to-solve stochastic optimization models in the snapshot problems, and we know of no real-world example where this stochastic replan technique has been applied.

More common is to add further constraints or penalty terms in the objective function to the snapshot problem of naive reoptimization. These constraints and penalties

represent a mostly heuristic safe-guard against unwanted system states that are not ruled-out by the offline-optimization model on known requests. One example is the common technique of using an average squared waiting time in the objective of the snapshot problem instead of an average waiting time [18, 16]. The goal of this is to balance the desire for individually not-too-large waiting times (fairness) with small average waiting times (performance). The main problem with this engineering approach is that in many cases nothing can be proved about the performance of the resulting replan policy.

Our application is an instance of the so-called *Online-dial-a-ride problem*. For the sake of an easier exposition, we restrict to the special case of a single server of capacity one. In our application, this means that want to control an idealized elevator of capacity one with one waiting queue of infinite capacity.

We show the following for such a system: the combination of ideas from online-optimization, model predictive control, and queuing theory yields experimentally and theoretically reliable control policies for these elevator systems. These stable policies are *not* defined as REPLAN-policies: our policies ignore some open-loop solutions and keep the ones computed earlier. The theoretical performance guarantees are dependent on a bound on the combinatorial load in the system: the *reasonability* of the input stream. As we go along, we put known results into context and present some so far unpublished results.

The chapter is structured as follows: Section 2 defines the problem under consideration formally. In Section 3 we define some online algorithms for the OLDARP, followed by some known performance results in Section 4. Our original contribution beyond these results is outlined in Section 5. The core concept dealt with in this chapter is introduced formally in Section 6, before we suggest the notion of strong stability in Section 7. Sections 8 and 9 present performance guarantees under reasonable load for two competitive algorithms IGNORE and SMARTSTART, respectively. Sections 10 and 11 show that such performance guarantees do not exist for the seemingly more natural competitive algorithms REPLAN and AVGFLOWREPLAN. The analysis for the new algorithm DELTAREPLAN is carried out in Section 12. Section 13 concludes the chapter with a conclusion and possible further directions.

2 Formal Problem Statement

After the informal introduction let us define the problem under consideration more precisely. We are given a metric space (X,d) with a special vertex $o \in X$ (the origin) in which a server of capacity $C \in \mathbb{R}_{\geq 0} \cup \{\infty\}$ moves at unit speed in order to serve transportation requests. 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. If r is a request, we also use $t(r)$ for its release time and $a(r)$ and $b(r)$ for its start end point, respectively.

A *transportation move* is a quadruple $m = (\tau,x,y,R)$, where a is the starting point and b the end point, and τ the starting time, while R is the set (possibly empty) of

requests carried by the move. The *arrival time* of a move is the sum of its starting time τ and $d(x,y)$. A *(closed) transportation schedule* for a sequence σ of requests is a sequence

$$S = (\tau_1, x_1, y_1, R_1), (\tau_2, x_2, y_2, R_2), \ldots, (\tau_\ell, x_\ell, y_\ell, R_\ell)$$

of transportation moves with the following properties:

(i) The $(i+1)$st move starts at the endpoint of the ith move and not earlier than the time that the ith move is completed, that is, $x_{i+1} = y_i$ and $\tau_{i+1} \geq \tau_i + d(x_i, y_i)$ for all i;
(ii) Each move carries at most C requests, that is, $|R_i| \leq C$ for all i;
(iii) For any request $r \in \sigma$, the subsequence of S consisting of those moves (τ_i, x_i, y_i, R_i) with $r \in R_i$ is a contiguous nonempty subsequence

$$S(r) = (\tau_l, x_l, y_l, R_l), \ldots, (\tau_{l+p}, x_{l+p}, y_{l+p}, R_{l+p})$$

of S which forms a transportation from $a(r)$ to $b(r)$, that is, $x_l = a(r)$ and $y_{l+p} = b(r)$. The sub-transportation $S(r)$ does not start before r is released, that is, $\tau_l \geq t(r)$.
(iv) The first move starts in the origin o and the last move ends in the origin o.

The time τ_1 and the point $x_1 \in X$ are called the *starting time* and the *starting point* of S. Similarly, the time $\tau_\ell + d(x_\ell, y_\ell)$ and the point y_ℓ are referred to as the *end time*, and the *end point* of S.

An *online algorithm* for OLDARP 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, see Condition (ii) above), while it does not know about requests that are presented 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. This is the core of *competitive analysis* of online algorithms.

An online algorithm A is called *c-competitive* if there exists a constant c such that for any finite request sequence σ the inequality $\mathsf{A}(\sigma) \leq c \cdot \mathsf{OPT}(\sigma)$ holds. Here, $\mathsf{X}(\sigma)$ denotes the objective function value of the solution produced by algorithm X on input σ and OPT denotes an optimal offline algorithm. Sometimes we are dealing with various objectives at the same time. We then indicate the objective *obj* in the superscript, as in $\mathsf{X}^{obj}(\sigma)$.

For a detailed set-up that focusses on competitive analysis see [1, 17].

3 Known Online Algorithms

Several online algorithms have been suggested in the literature so far: we discuss REPLAN, IGNORE, and SMARTSTART because our stability results refer to them. All these algorithms stem from [1]. The algorithm REPLAN is based on ideas in [3]; the algorithm IGNORE appears in [21] in a more general context. The algorithms can be considered as typical representatives of construction principles for online algorithms: REPLAN reoptimizes whenever a new request arrives, IGNORE reoptimizes only when it becomes idle whereafter it immediately continues to work, SMARTSTART reoptimizes only when idle and stays idle deliberately for a certain amount of time to gather more information about unserved requests.

The online algorithm REPLAN for the OLDARP is based on the general idea of a replan algorithm in the introduction in Section 1.

Definition 1 (Algorithm REPLAN). Whenever a new request becomes available, REPLAN computes a preliminary transportation schedule for the set R of all available requests by solving the problem of minimizing the total completion time of R. Then it moves the server according to that schedule until a new request arrives or the schedule is done.

The online algorithm IGNORE makes full use of every schedule it computes before it recomputes a new schedule:

Definition 2 (Algorithm IGNORE). Algorithm IGNORE works with an internal buffer. It may assume the following states (initially it is IDLE):

IDLE Wait for the next point in time when requests become available. Goto PLAN.

BUSY While the current schedule is in work store the upcoming requests in a buffer ("ignore them"). Goto IDLE if the buffer is empty else goto PLAN.

PLAN Produce a preliminary transportation schedule for all currently available requests R (taken from the buffer) minimizing the total completion time *comp* of R. (Note: This yields a feasible transportation schedule for R because all requests in R are immediately available.) Goto BUSY.

The algorithm SMARTSTART was developed to improve the competitive ratios of REPLAN and IGNORE. The idea of this algorithm is basically to emulate the IGNORE algorithm but to make sure that each sub-transportation schedule is completed "not too late": if a sub-schedule would take "too long" to complete then the algorithm waits for a specified amount of time. Intuitively this construction tries to avoid the worst-case situation for IGNORE where, right after the algorithm starts its work on a schedule, a new request becomes known.

In this section we use $l(S)$ to denote the length of a schedule (tour) S computed for a (sub-) set of requests. SMARTSTART has a fixed "waiting scaling" parameter $\theta > 1$. From time to time the algorithm consults its "work-or-sleep" routine: this subroutine computes an (approximately) shortest schedule S for all unserved requests, starting and ending in the origin. If this schedule can be completed no

later than time θt, i.e., if $t + l(S) \leq \theta t$, where t is the current time and $l(S)$ denotes the length of the schedule S, the subroutine returns (S, work), otherwise it returns (S, sleep).

In the sequel it will be convenient again to assume that the "work-or-sleep" subroutine uses a ρ-approximation algorithm for computing a schedule: the approximation algorithm always finds a schedule of length at most ρ times the optimal one.

Definition 3 (Algorithm SMARTSTART). The server of algorithm SMARTSTART can assume three states (initially it is IDLE):

IDLE If the algorithm is idle at time T and new requests arrive, it calls "work-or-sleep". If the result is (S, work), the algorithm enters the busy state where it follows schedule S. Otherwise the algorithm enters the sleeping state with wakeup time t', where $t' \geq T$ is the earliest time such that $t' + l(S) \leq \theta t'$ and $l(S)$ denotes the length of the just computed schedule S, i.e., $t' = \min\{ t \geq T : t + l(S) \leq \theta t \}$.

SLEEPING In the sleeping state the algorithm simply does nothing until its wakeup time t'. At this time the algorithm reconsults the "work-or-sleep" subroutine. If the result is (S, work), then the algorithm enters the busy state and follows S. Otherwise the algorithm continues to sleep with new wakeup time $\min\{ t \geq t' : t + l(S) \leq \theta t \}$.

BUSY In the busy state, i.e, while the server is following a schedule, all new requests are (temporarily) ignored. As soon as the current schedule is completed the server either enters the idle-state (if there are no unserved requests) or it reconsults the "work-or-sleep" subroutine which determines the next state (SLEEPING or BUSY).

4 Known Performance Guarantees

Competitive analysis of OLDARP provided the following (see [1]):

- IGNORE and REPLAN are 2.5-competitive for the goal of minimizing the *total completion time* of the schedule; SMARTSTART is 2-competitive for this problem, which is best-possible.
- For the task of minimizing the *maximal (average) waiting time* or the *maximal (average) flow time* there can be no algorithm with constant competitive ratio. In particular, the algorithms REPLAN, IGNORE, and SMARTSTART have unbounded competitive ratios for this problem.

It should be noted that the corresponding offline-versions with release times (where all requests are known at the start of the algorithm) are NP-hard to solve for the objective functions of minimizing the average or maximal flow time—it is even NP-hard to find a solution within a constant factor from the optimum [15]. The offline-version without release times of minimizing the total completion time is polynomially solvable on special graph classes but NP-hard in general [9, 2, 8, 13].

If we are considering a continuously operating system with continuously arriving requests (i.e., the request set may be infinite) then the total completion time unbounded anyway, thus meaningless. Thus, in this case, the existing positive results cannot be applied and the negative results tell us that we cannot hope for performance guarantees that may be relevant in practice. In particular, the performances of the two algorithms REPLAN and IGNORE cannot be distinguished by classical competitive analysis at all (both are 2.5 competitive w.r.t. the total completion time and not competitive at all w.r.t. the average or maximal flow time), and the performance of SMARTSTART can not be distinguished from any other algorithm if the average or maximal flow time is the goal.

In order to find theoretical guidance which algorithm should be chosen, the notion of Δ-*reasonable load* was developed [12]. A set of requests is Δ-reasonable if requests released during a period of time $\delta \geq \Delta$ can always be served in time at most δ. A set of requests R is *reasonable* if there exists a $\Delta < \infty$ such that R is Δ-reasonable. That means, for non-reasonable request sets we find arbitrarily large periods of time where requests are released faster than they can be served—even if the server has an optimal offline schedule and all requests can be served immediately. When a system has only to cope with reasonable request sets, we call this situation *reasonable load*. Section 6 is devoted to the exact mathematical setting of this idea, because we need it for the new results.

The main historical result based on this idea in [12] is: For the OLDARP under Δ-reasonable load, IGNORE yields a maximal and an average flow time of at most 2Δ, whereas the maximal and the average flow time of REPLAN are unbounded. The algorithms IGNORE and REPLAN have to solve a number of offline instances of OLDARP, which is in general NP-hard, as we already remarked. One can derive results for IGNORE when using an approximate algorithm for solving offline instances of OLDARP (for approximation algorithms for offline instances of OLDARP, refer to [9, 2, 8, 13]). To this end, the notion of reasonable request sets was refined [12], introducing a second parameter that tells us how "fault tolerant" the request set is. In other words, the second parameter tells us, how "good" the algorithm has to be to show stable behavior. Again, roughly speaking, a set of requests is (Δ,ρ)-reasonable if requests released during a period of time $\delta \geq \Delta$ can be served in time at most δ/ρ. If $\rho = 1$, we get the notion of Δ-reasonable as described above. For $\rho > 1$, the algorithm is allowed to work "sloppily" (e.g., employ approximation algorithms) or have break-downs to an extent measured by ρ and still show a stable behavior.

5 Outline of New Contributions

Simulation results [11] show that IGNORE indeed outperforms REPLAN in terms of the maximal flow time, but in terms of the average flow time the behaviour of REPLAN is usually much better. This left open the question about whether IGNORE can be improved empirically without losing the performance guarantee. Alternatively: is there a version of REPLAN that wins the performance guarantee of IGNORE but

stays empirically efficient? As an answer to this question we present the algorithm DELTAREPLAN in Section 12.

The following results in this chapter have not been published elsewhere before:

- We present a proof that the replan policy SMARTSTART that is optimally competitive for the total completion time (the *makespan*) has bounded flow times under reasonable load as well;
- we show an example for which a replan policy with snapshot objective "minimize the average flow time" produces unbounded maximal and average flow times in the long run;
- we present one particular replan policy DELTAREPLAN that inherits the performance guarantee of IGNORE but is able to yield a better average flow time in simulations;
- we show that using a policy with bounded flow times yields uniformly bounded waiting queues, i.e., strong stability.

6 Reasonable Load in Detail

Crucial for the concept of reasonable load is the offline version of a request set.

Definition 4. The *offline version* of $r = (t, a, b)$ is the request

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

The *offline version* of R is the request set

$$R^{\mathit{offline}} := \left\{ r^{\mathit{offline}} \;:\; r \in R \right\}.$$

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

Definition 5. Let R be a finite request set for OLDARP. The *release span* $\delta(R)$ of R is defined as

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

Provably good offline-algorithms exist for the total completion time 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. The idea is stolen from queuing theory where the input rate should not exceed the output rate. In the following we propose a mathematical set-up which 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 6. Let R be a request set for the OLDARP. A weakly monotone function

$$f\colon \begin{cases} \mathbb{R} \to \mathbb{R}, \\ \delta \mapsto f(\delta); \end{cases}$$

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 completion time $\mathsf{OPT}^{comp}(S^{offline})$ of the optimum schedule for the offline version $S^{offline}$ of S is at most $f(\delta)$. In formula:

$$\mathsf{OPT}^{comp}(S^{offline}) \leq f(\delta).$$

Remark 1. If the whole request set R is finite then there is always the trivial load bound given by the total completion time of R. For every load bound f we may set $f(0)$ to be the maximum completion time we need for a single request, and nothing better can be achieved.

A "stable" situation would easily obtained by a load bound equal to the identity $x \mapsto x$ on $\mathbb{R}$. (By "stable" we mean that the number of unserved requests in the system does not become arbitrarily large.) In that case we would never get more work to do than we can accomplish. If it has a load bound equal to a function id/ρ, where id is the identity and where $\rho \geq 1$, then ρ measures the tolerance of the request set: assume we have an offline-algorithm at our disposal that produces in the worst case cost of ρ times the cost of an optimal offline algorithm, then we can still accomplish all the incoming work by using the IGNORE-algorithm: compute a ρ-approximate schedule for the set R of all released but unserved requests. The load bound and the performance guarantee ensure that the schedule takes no longer than $\rho \cdot \Delta(R)/\rho = \Delta(R)$. Thus, the set of requests that are released in the meantime has a release span no larger than $\Delta(R)$, and we can proceed by computing a ρ-approximate schedule for that set.

However, we cannot expect that the identity (or any linear function) is a load bound for OLDARP 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 7. A load bound f is *(Δ,ρ)-reasonable* for some $\Delta, \rho \in \mathbb{R}$ with $\rho \geq 1$ if

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

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.

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

Remark 2. 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 policy is good enough 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.

In a sense, Δ is a measure for the combinatorial difficulty of the request set R. If R is not Δ-reasonable for any $\Delta > 0$, then this indicates that the capacity of the system does not suffice to keep the system stable. Then, the task is not to find the best control but to add capacity first.

Thus, it is natural to ask for performance guarantees for the flow times of algorithms in terms of the reasonability Δ of the input. This is discussed for various algorithms in Sections 8 through 12. Before that, we want to argue that flow time bounds guarantee a certain form of stability.

7 Strong Stability

We want to find an online algorithm for which there is a uniform bound on the number of requests in the system. More formally:

Definition 8. An online algorithm ALG for OLDARP on (X,d) with origin o is *strongly Δ-stable* if there exists $M \geq 0$ such that for each Δ-reasonable request set R the number of unserved requests in the system controlled by ALG is never larger than M.

In the stochastic setting, Little's formula (see, e.g., [10]) provides a relation between the traffic, the expected number of requests in the system and the expected flow time of the requests: the expected number of requests in the system equals the average number of requests entering the system times the expected flow time of requests. We would like to replace the traffic intensity by our Δ, but since we are in a worst-case setting, the corresponding relation does not always hold.

In contrast to traffic conditions in queuing theory, Δ is only indirectly related to the number of requests in the system. The problem occurs when the service times for requests are not bounded away from zero. The extreme case is an Online Traveling Salesman Problem where requests must be visited, nothing else; in that case, the server can serve an unlimited number of requests in arbitrarily little time if it is already very close to the position of the requests. For a short time then, there may be an unlimited number of requests in the system although serving them requires only an arbitrarily small amount of time, thereby not violating any Δ-reasonability requirement. It is clear that in such a situation no algorithm can achieve strong stability.

The situation is different when serving a request takes at least time $\tau > 0$. In the elevator example this is true, because each pallet has to be transported for at least one floor. We call this variant of OLDARP the *Online-Dial-a-Ride-Problem with minimal transport time* τ or τ-OLDARP, for short.

We then obtain the following:

Theorem 1. *If the maximal flow time of* ALG *for* τ-OLDARP *is at most* $f(\Delta)$ *for all* Δ*-reasonable request sets, then* ALG *is strongly* Δ*-stable. More specifically, the number of requests in the system is never larger than* $f(\Delta)/\tau$.

Proof. The time we need to serve a request is at least τ. If a request subset with release span at most Δ contains more than Δ/τ requests, then we need more time than $\Delta = \tau\Delta/\tau$ to serve it offline. The maximal number of requests that can enter the system in time Δ is therefore Δ/τ. If each request leaves the system after at most $f(\Delta)$ time units, then there may be at most

$$f(\Delta) \cdot \frac{\Delta}{\tau} \cdot \frac{1}{\Delta} \tag{1}$$

requests at the same time in the system. □

The result of this (elementary) discussion is: in order to obtain strongly stable online algorithms it is sufficient to find online algorithms with bounded maximal flow times.

8 Bounds for the Flow Times of IGNORE

We are now in a position to prove bounds for the maximal resp. average flow time in the OLDARP for algorithm IGNORE. We assume that IGNORE solves offline instances of OLDARP employing a ρ-approximation algorithm.

Let us consider the intervals in which IGNORE organizes its work in more detail. The algorithm IGNORE induces a dissection of the time axis $\mathbb{R}$ in the following way: We can assume, w.l.o.g., that the first set of requests arrives at time 0. Let $\delta_0 = 0$, i.e., the point in time where the first set of requests is released (these are processed by IGNORE in its first schedule). For $i > 0$ let δ_i be the duration of the time period the server is working on the requests that have been ignored during the last δ_{i-1} time units. Then the time axis is split into the intervals

$$[\delta_0 = 0, \delta_0], (\delta_0, \delta_1], (\delta_1, \delta_1 + \delta_2], (\delta_1 + \delta_2, \delta_1 + \delta_2 + \delta_3], \ldots$$

Let us denote these intervals by $I_0, I_1, I_2, I_3, \ldots$. Moreover, let R_i be the set of those requests that are presented in I_i. Clearly, the complete set of requests R is the disjoint union of all the R_i.

At the end of each interval I_i we solve an offline problem: all requests to be scheduled are already available. The work on the computed schedule starts immediately

(at the end of interval I_i) and is done δ_{i+1} time units later (at the end of interval I_{i+1}). On the other hand, the time we need to serve the schedule is not more than ρ times the optimal completion time $\mathsf{OPT}^{comp}(R_i^{offline})$ of $R_i^{offline}$. In other words:

Lemma 1. *For all $i \geq 0$ we have*

$$\delta_{i+1} \leq \rho \cdot \mathsf{OPT}^{comp}(R_i^{offline}).$$

Let us now state and prove the main result of this section, first proved in [12], about the maximal flow time $\mathsf{IGNORE}^{maxflow}(R)$ incurred by IGNORE on any reasonable request set R.

Theorem 2 ([12]). *Let $\Delta > 0$ and $\rho \geq 1$. For all instances of* OLDARP *with (Δ,ρ)-reasonable request sets,* IGNORE *employing a ρ-approximate algorithm for solving offline instances of* OLDARP *yields a maximal flow time of no more than 2Δ.*

Proof. Let r be an arbitrary request in R_i for some $i \geq 0$, i.e., r is released in I_i. By construction, the schedule containing r is finished at the end of interval I_{i+1}, i.e., at most $\delta_i + \delta_{i+1}$ time units later than r was released. Thus, for all $i > 0$ we get that

$$\mathsf{IGNORE}^{maxflow}(R_i) \leq \delta_i + \delta_{i+1}.$$

If we can show that $\delta_i \leq \Delta$ for all $i > 0$ then we are done. To this end, let $f : \mathbb{R} \to \mathbb{R}$ be a (Δ,ρ)-reasonable load bound for R. Then $\mathsf{OPT}^{comp}(R_i^{offline}) \leq f(\delta_i)$ because $\delta(R_i) \leq \delta_i$.

By Lemma 1, we get for all $i > 0$

$$\delta_{i+1} \leq \rho\,\mathsf{OPT}^{comp}(R_i^{offline}) \leq \rho f(\delta_i) \leq \max\{\delta_i, \Delta\}.$$

Using $\delta_0 = 0$ the claim now follows by induction on i. □

The average flow time of IGNORE is also bounded, because the average is never larger than the maximum.

Corollary 1. *Let $\Delta > 0$. For all Δ-reasonable request sets algorithm* IGNORE *yields a average flow time of no more than 2Δ.*

9 Bounds for the Flow Times of SMARTSTART

The analysis of SMARTSTART under reasonable load was not published before; it essentially parallels that of IGNORE, so we only highlight the differences. The crucial observation needed is formulated in the following lemma:

Lemma 2. *For (Δ,ρ)-reasonable request sequences the server of* SMARTSTART *never sleeps after time $\bar{t} := \frac{\Delta}{\theta - 1}$.*

Proof. Consider a call to the "work-or-sleep" routine at an arbitrary time $t \geq \bar{t}$. Let R be the set of requests not served by SMARTSTART at time t and let S be a ρ-approximate shortest schedule for R. By the (Δ,ρ)-reasonability of the input sequence, the length of schedule S for R can be bounded from above by

$$l(S) \leq \rho \cdot \max\left\{\frac{\Delta}{\rho}, \frac{\delta(R)}{\rho}\right\} = \max\{\Delta, \delta(R)\}.$$

Trivially, we have $\delta(R) \leq t$, since all requests in R have been released at time t. Hence, it follows that

$$\begin{aligned} t + l(S) &\leq t + \max\{\Delta, \delta(R)\} \\ &\leq t + \max\{\Delta, t\} && \text{(since } \delta(R) \leq t\text{)} \\ &= t + \max\{(\theta-1)\bar{t}, t\} && \text{(since } \bar{t} = \Delta/(\theta-1)\text{)} \\ &\leq \theta t && \text{(since } t \geq \bar{t}\text{).} \end{aligned}$$

Consequently, the "work-or-sleep" routine does not return the invitation to sleep.

The same arguments as given above show that, if SMARTSTART goes to sleep before some time $t < \bar{t}$, the wakeup time is no later than time $\bar{t}$. Hence, the lemma follows. □

Let S be the last schedule started by SMARTSTART no later than time $\bar{t}$ and denote by $t_S \leq \bar{t}$ its start time. From Lemma 2 we conclude that from time $\bar{t}$ on, SMARTSTART behaves like IGNORE, provided the input sequence is (Δ,ρ)-reasonable. Using the arguments given in the proof of Theorem 2 we can conclude that the flow time of any request released after time t_S is bounded from above by 2Δ.

It remains to treat the requests released before time $\bar{t}$. Using again the arguments of Theorem 2 we derive that all requests released after time t_S have flow time at most 2Δ and we finally need to consider those requests released until time t_S. Each of these requests is either served by S or by an even earlier schedule. Since by definition of SMARTSTART, the transportation schedule S is completed no later than time $\theta t_S < \theta\bar{t} = \frac{\theta}{\theta-1}\Delta$, we obtain the following result:

Theorem 3. *Let $\Delta > 0$ and $\rho \geq 1$. For all instances with (Δ,ρ)-reasonable request sets, Algorithm* SMARTSTART *employing a ρ-approximation algorithm in its "work-or-sleep" routine yields a maximal flow time of no more than* $\max\left\{\frac{\theta}{\theta-1}\Delta, 2\Delta\right\}$. *In particular, if $\theta \geq 2$, then the maximal flow time provided by* SMARTSTART *is bounded from above by 2Δ.* □

As in the case of IGNORE a we can derive a trivial upper bound of 2Δ for the average flow time of SMARTSTART under reasonable load.

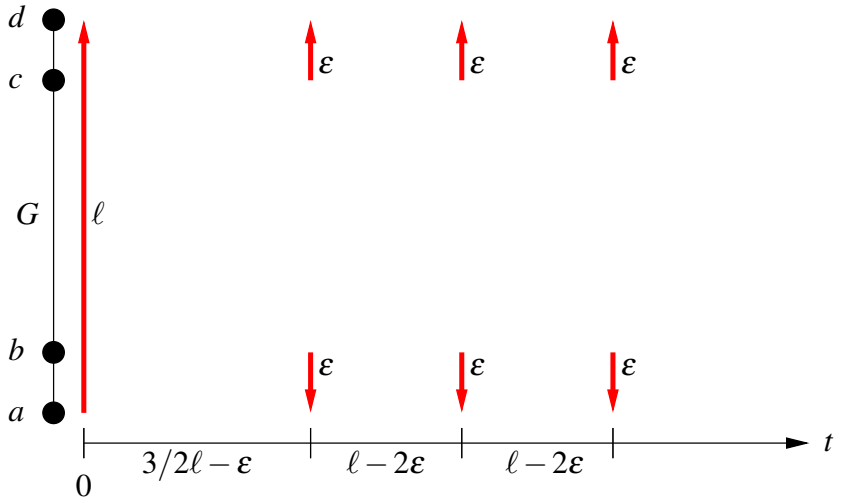

Fig. 2 A sketch of a $(2\frac{2}{3}\ell)$-reasonable instance of OLDARP ($\ell = 18\varepsilon$). The horizontal axis holds the time, the vertical axis depicts the metric space in which the server moves. A request is denoted by an arrow from its starting point to its end point horizontally positioned at its release time.

10 An Example with Unbounded Flow Times for REPLAN

In the sequel, we provide an instance of OLDARP and a Δ-reasonable request set R such that the maximal flow time $\text{REPLAN}^{maxflow}(R)$ (and thus also the average flow time) of REPLAN is unbounded. This was first proved in [12]. Recall that REPLAN uses a snapshot optimization problem in which the total completion time is minimized. Hence, REPLAN is not a naive replan policy since our evaluation objective is the maximal flow time.

Theorem 4 ([12]). *There is an instance of* OLDARP *under reasonable load such that the maximal and the average flow time of* REPLAN *is unbounded.*

Proof. In Figure 2 there is a sketch of an instance for the OLDARP. The metric space is a path on four nodes a,b,c,d with origin a; the length of the path is ℓ, the distances are $d(a,b) = d(c,d) = \varepsilon$, and hence $d(b,c) = \ell - 2\varepsilon$. At time 0 a request from a to d is issued; at time $3/2\ell - \varepsilon$, the remaining requests periodically come in pairs from b to a and from c to d, resp. The time distance between them is $\ell - 2\varepsilon$.

We show that for $\ell = 18\varepsilon$ the request set R indicated in the picture is $2\frac{2}{3}\ell$-reasonable. Indeed: it is easy to see that the first request from a to d does not influence reasonability. Consider an arbitrary set R_k of k adjacent pairs of requests from b to a resp. from c to d. Then the release span $\delta(R_k)$ of R_k is

$$\delta(R_k) = (k-1)(\ell - 2\varepsilon).$$

The offline version ${R_k}^{offline}$ of R_k can be served as follows: first, move the server to c, the starting point of the upper requests: this contributes cost $\ell - \varepsilon$. Next, serve all the upper requests and go back to c: this contributes cost $k \times 2\varepsilon$. Then, go down to b, the starting point of the lower requests: this contributes another $\ell - 2\varepsilon$ to the cost. Now, serve the first lower requests: the additional cost for this is ε. Finally, serve the remaining lower requests at an additional cost of $(k-1) \cdot 2\varepsilon$. In total, we have the following:

$$\mathsf{OPT}^{comp}({R_k}^{offline}) = 2\ell + (k-1) \cdot 4\varepsilon.$$

In order to find the smallest parameter Δ for which the request set R_k is Δ-reasonable we solve for the integer $k-1$ and get

$$k - 1 = \left\lceil \frac{2\ell}{\ell - 6\varepsilon} \right\rceil = 3.$$

Hence, we can set Δ to

$$\Delta := \mathsf{OPT}^{comp}({R_4}^{offline}) = 2\tfrac{2}{3}\ell.$$

Now we define

$$f : \begin{cases} \mathbb{R} \to \mathbb{R}, \\ \delta \mapsto \begin{cases} \Delta & \text{for } \delta < \Delta, \\ \delta & \text{otherwise.} \end{cases} \end{cases}$$

By construction, f is a load bound for R_4. Because the time gap after which a new pair of requests occurs is certainly larger than the additional time we need to serve it (offline), f is also a load bound for R. Thus, R is Δ-reasonable, as desired.

Now: how does REPLAN perform on this instance? In Figure 3 we see the track of the server following the preliminary schedules produced by REPLAN on the request set R.

The maximal flow time of REPLAN on this instance is realized by the flow time of the request $(3/2\ell - \varepsilon, b, a)$, which is unbounded.

Moreover, since all requests from b to a are postponed after serving all the requests from c to d we get that REPLAN produces an unbounded average flow time as well. □

In Figure 4 we show the track of the server under the control of the IGNORE-algorithm. After an initial inefficient phase the server ends up in a stable operating mode. This example also shows that the analysis of IGNORE in Section 8 is sharp.

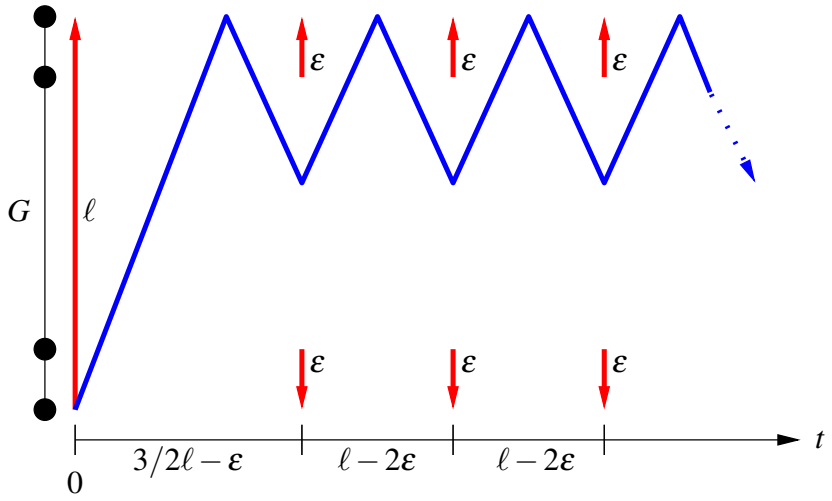

Fig. 3 The track of the REPLAN-server is drawn as a line in the diagram: at each point in time t we can read off the position of the server by looking at the height of the line at the horizontal position t. Because a new pair of requests is issued exactly when the server is still closer to the requests at the top all the requests at the bottom will be postponed in an optimal preliminary schedule. Thus, the server always returns to the top when a new pair of requests arrives.

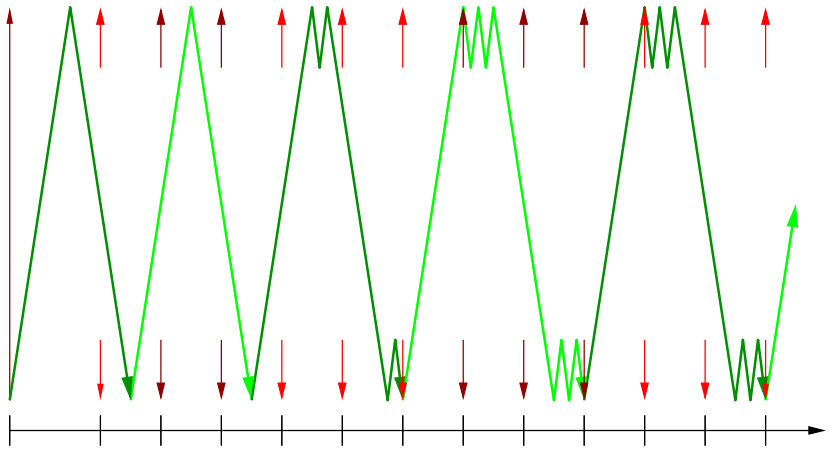

Fig. 4 The track of the IGNORE-server.

11 An Example with Unbounded Flow Times for AVGFLOWREPLAN

It is quite a natural question to ask whether modified replan strategies AVGFLOWREPLAN or MAXFLOWREPLAN that use snapshot problems that minimize the average resp. maximal flow times would give a reasonable bound on the maximal and average flow times in the online situation. In our taxinomy, MAXFLOWREPLAN implements the naive replan policy when the evaluation objective is the minimization of the maximal flow time. And AVGFLOWREPLAN corresponds to the naive replan policy when the evaluation objective is the minimization of the average flow time.

We mentioned already that the offline problem of minimizing the average flow time is very hard. In the offline problem that AVGFLOWREPLAN has to solve, however, all requests have release times in the past. It is then easy to see that the problem is equivalent to the minimization of the average completion time counted from the point in time where the planning takes place. Moreover, since the average flow time is larger by the "average age" of the requests, the performance guarantees of approximation algorithms minimizing the average completion time carry over. Still, in our computational experience minimization of the average completion time takes more time than minimizing the total completion time.

Anyway: the following result shows that even under reasonable load we cannot expect a worst case stable behaviour of AVGFLOWREPLAN, a so far unpublished result.

Theorem 5. *There is an instance of* OLDARP *under reasonable load such that the maximal and average flow times of* AVGFLOWREPLAN *are unbounded.*

Proof. We construct a set of requests in the same metric space as in the previous Section 10 as follows:

- At time 0 we issue again one request from a to d.
- At time $T_0 := 3/2\ell - \varepsilon$ we issue a pair of requests R_1^u from c to d and R_1^l from b to a.
- At time $T_{i+1} := T_i + \ell + 2(i-2)\varepsilon$ we issue
  - a set of i "upper" requests R_{i+1}^u from c to d and
  - one "lower" request R_{i+1}^l from b to a.

Figure 5 sketches the construction.

For $\ell = 18\varepsilon$ this request set is again $2\frac{2}{3}\ell$-reasonable since we have increased the time intervals between the release times of the requests by the additional amount that is needed to serve the additional copies of upper requests.

At time T_i, for all $i > 0$, AVGFLOWREPLAN has still to serve as many upper requests as there are lower requests. Thus, at T_i the schedule with minimum average flow time for the currently available requests serves the upper requests first. Hence, the requests at the bottom have to wait for an arbitrarily long period of time.

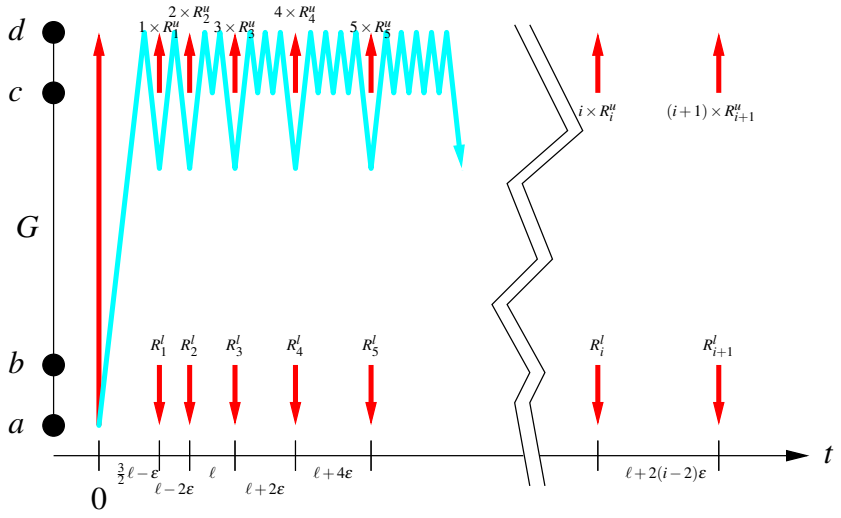

Fig. 5 The track of the AVGFLOWREPLAN-server on a the example from Theorem 5.

In order to prove the assertion concerning the average flow time we consider the result $f(R_N)$ that AVGFLOWREPLAN produces on the input set R_N that contains all requests up to time T_N.

The sum of all flow times $f_\Sigma(R_N)$ is dominated by the waiting times of the lower requests. That is, it is at least

$$\begin{aligned} f_\Sigma(R_N) &\geq \sum_{k=1}^{N}\sum_{i=k}^{N}(\ell+2(i-2)\varepsilon) \\ &\geq \sum_{k=1}^{N}\sum_{i=k}^{N}(i-2)\varepsilon. \end{aligned}$$

The number of requests $\#R_N$ in R_N is

$$\#R_N = 1+\sum_{k=1}^{N}(k+1),$$

so that

$$f(R_N) = \frac{f_\Sigma(R_N)}{\#R_N} \xrightarrow{N\to\infty} \infty,$$

which completes the proof. □

A policy that minimizes just the maximal flow time does not make a lot of sense since sometimes this only determines which request is to be served first; the order in which all the other requests are scheduled is unspecified. Thus, the most sensible policy in this respect seems to be the following: consider an offline instance of the

dial-a-ride problem. The vector consisting of all flow times of requests in a feasible solution ordered decreasingly is the *flow vector*. All flow vectors are ordered lexicographically. The online policy MAXFLOWREPLAN for the online dial-a-ride problem does the following: whenever a new request becomes available MAXFLOWREPLAN computes a new schedule of all yet unserved requests minimizing the flow vector.

It is an open problem what the performance of this policy is under Δ-reasonable load. In practice, however, it is probably too difficult to solve the snapshot problem with this objective function.

12 Combining the Best of two Ideas: DELTAREPLAN

A closer inspection of the behaviour of IGNORE and SMARTSTART, resp., versus the behaviour of REPLAN and AVGFLOWREPLAN, resp., shows: REPLAN is unstable under Δ-reasonable load because of infinite deferment of requests, which can not happen in IGNORE, since IGNORE does not replan often enough to defer requests. On the other hand: reoptimizing less frequently means leaving out opportunities to improve, and thus, on average, IGNORE is empirically worse than REPLAN. The key idea to combine the advantages of both policies is to constrain the reoptimization that REPLAN performs. The result is the following online algorithm DELTAREPLAN, so far unpublished, which works as follows:

Whenever a new request becomes available, DELTAREPLAN computes a preliminary transportation schedule for the set R of all available requests by solving the problem of minimizing the total completion time of $R^{offline}$ under the restriction that no request in the transportation schedule has predicted flow time more than 2Δ. If the makespan of the optimal transportation schedule is at most Δ, the new schedule is accepted and becomes the active schedule. The new schedule is rejected otherwise, whence the previous schedule is kept active. It then moves the server according to the active schedule until a new request arrives or the schedule is done. Note, that the new requests that trigger the reoptimization are not rejected. It is the new schedule that is rejected. Thus, since we do not allow rejection of requests, DELTAREPLAN is only feasible if each request is in an accepted schedule, sooner or later.

Summarized, we define:

Definition 9 (Algorithm DELTAREPLAN). Algorithm DELTAREPLAN (Δ, ρ) has parameters $\Delta > 0, \rho > 1$ (indicating that it aims at (Δ, ρ)-reasonable request sets) and works with an internal buffer holding an active schedule and possibly some requests. It may assume the following states (initially it is IDLE):

IDLE Wait for the next point in time when requests become available. Goto PLAN.

PLAN Produce a preliminary transportation schedule for all currently available requests R (taken from the buffer) minimizing *comp* for $R^{offline}$ under the constraint that no request has a predicted flow time exceeding 2Δ, possibly by a

ρ-approximation algorithm. If the problem is infeasible or the computed completion time exceeds Δ reject the new schedule and keep the old one active, thereby buffering the new requests. Otherwise replace the active schedule by the new one. Goto BUSY.

BUSY Serve requests according to the active schedule. If a new requests is released or the active schedule is done, goto PLAN.

The result is:

Theorem 6. *Let $\Delta > 0$ and $\rho \geq 1$. For all instances with (Δ, ρ)-reasonable request sets, Algorithm* DELTAREPLAN *(Δ, ρ) employing a ρ-approximation algorithm for reoptimization yields a maximal flow time of no more than 2Δ.*

Proof. As long as all new schedules are rejected, DELTAREPLAN (Δ, ρ) works in the same way as IGNORE. Whenever a new schedule is accepted the constraints on the flow times of the scheduled requests guarantee the bound by construction. Since no schedule of length larger than Δ is accepted, rejection of all optimal schedules thereafter yields a maximal release span for buffered requests of at most Δ. The buffered requests can therefore theoretically be served in time at most Δ/ρ. Because DELTAREPLAN(Δ, ρ) employs a ρ-approximation algorithm, it computes a schedule of length at most Δ. Since all requests during the work on a schedule have been ignored, the flow times of them are exactly the flow times IGNORE would have produced. Thus, the flow time constraints are satisfied for all of them. Therefore, the first computed schedule after the work on the active schedule has finished will be accepted. Consequently, every request will be in an accepted schedule at some point. Thus, the claim holds. □

What happens if we do not know, how reasonable the request sets are going to be, i.e., if we do not know (Δ, ρ) in advance? Let us restrict to the case with approximation factor $\rho = 1$ in order to concentrate on the core aspect. If DELTAREPLAN is run with a $\Delta' < \Delta$ on a Δ-reasonable request set then still all schedules that would be rejected with DELTAREPLAN (Δ) would also be rejected by DELTAREPLAN (Δ'). A problem may occur that when the active schedule is done, the new schedule has makespan larger than Δ' so that we have to reject it; but then we are stuck. We can then modify DELTAREPLAN in three ways to by-pass this problem:

IGNORE-DELTAREPLAN Accept all schedules that are computed because the old schedule is done.

DOUBLE-DELTAREPLAN Take $\Delta'' := 2\Delta'$ as a new estimate of Δ and run DELTAREPLAN (Δ''). This is often called *doubling technique* for parametrized algorithms [4].

DELTAREPLAN Take the makespan Δ'' of the new schedule (which is at most Δ) as a new estimate of Δ and run DELTAREPLAN (Δ'').

The first option uses IGNORE as a back-up whenever DELTAREPLAN (Δ') fails to produce a schedule. This way, we obtain the same bound 2Δ on the flow times but we may lose some efficiency due to too many rejected schedules.

Theorem 7. *Let $\Delta > 0$ and $\rho \geq 1$. For all instances with (Δ, ρ)-reasonable request sets, Algorithm* IGNORE-DELTAREPLAN *employing a ρ-approximation algorithm for reoptimization yields a maximal flow time of no more than 2Δ.* □

The estimate for Δ in the doubling technique will at some point surpass the true Δ. Then, we still get a bound on the flow times, but only with respect to the overestimated Δ, i.e., a bound of 4Δ in the worst case.

Theorem 8. *Let $\Delta > 0$ and $\rho \geq 1$. For all instances with (Δ, ρ)-reasonable request sets, Algorithm* DOUBLE-DELTAREPLAN *employing a ρ-approximation algorithm for reoptimization yields a maximal flow time of no more than 4Δ.* □

Since for DELTAREPLAN the estimates for Δ never exceed Δ and the reoptimization problems as well as the acceptance of new schedules are at least as constrained as for DELTAREPLAN(Δ), we conclude that DELTAREPLAN has flow times bounded by 2Δ, and the loss of efficiency is decreasing as the estimate of Δ gets closer and closer to Δ. We obtain the following result:

Theorem 9. *Let $\Delta > 0$ and $\rho \geq 1$. For all instances with (Δ, ρ)-reasonable request sets, Algorithm* DELTAREPLAN *employing a ρ-approximation algorithm for reoptimization yields a maximal flow time of no more than 2Δ.* □

This basic DELTAREPLAN-technique can be applied in much more general situations (see [20] for a sketch). We arrived at an algorithm very much in the spirit of MPC with ingredients from online-optimization and queuing theory: For a classical problem in online optimization, estimate the characteristic difficulty of the input stream in terms of Δ, the definition of which was inspired by queuing theory, and use cleverly constrained reoptimization model with a suitable objective to obtain a strongly stable system.

13 Conclusion

We have shown how naive reoptimization policies in the control of elevators may lead to unstable systems. Moreover, via the notion of (Δ, ρ)-reasonable load we found a modification of the usual reoptimization policies that achieves strong stability, a new notion aiming at stability in worst-case analysis in a queuing system. The new notions and the policies emerge as a combination of paradigms from basic online-optimization, queuing theory, and model predictive control. We conjecture that closing the gap between these fields will lead to interesting, sometimes surprisingly simple but yet useful innovations.

The analysis under reasonable load is valid in much larger generality. Essentially, every system in which servers have to serve requests can be captured. This encompasses also general dial-a-ride problems. A generic formulation of the principle based on a generic integer linear programming formulation of the offline version of some online problem is presented in [20]. We did not present this here for the sake of a less abstract exposition.

There are a couple of open questions in this area:

- Does MAXFLOWREPLAN produce bounded flow times in terms of Δ under Δ-reasonable load?
- The policies in this chapter are all based on the computation of higher-level information, namely a precomputed schedule. On this higher level, there is no immediate notion of a "terminal state". Is there any version of "terminal state constraints" or "terminal costs" for the snapshot problem that can guarantee stability of the corresponding replan policy?
- Of course, since the reasonability Δ is a worst-case measure, performance may benefit if Δ is considered as a dynamically changing property of the request set which should be estimated in a time-dependent fashion in order not to use a too large Δ most of the time; especially, when there are traffic peaks. Can one rigorously quantify the benefits of such a dynamic approach?
- We have no non-trivial theoretical guarantees for the expected average flow-times over a distribution of request sets. Does DELTAREPLAN have provably better average flow times than IGNORE, as it seems empirically?
- Experience shows that minimizing the average *quadratic* flow times in the snapshot problem leads to empirically stable systems. Can one guarantee strong stability for them?

The LCCC theme semester revealed that quite a few types of logistic control problems are attacked by more than one mathematical community; up to now rather in isolation than in cooperation. We would be very happy if this volume—and, in particular, this chapter—motivated a thorough performance comparison. More specifically: what can be achieved, in theory and practice, by the various techniques in queuing theory, model predictive control, stochastic dynamic optimization, and online optimization on *a common set of problems*?

*

We thank two anonymous referees for helpful comments on the presentation of this chapter. The second author is grateful for the opportunity to participate in a very inspiring theme semester at LCCC and the financial support by LCCC.

References

1. Ascheuer, N., Krumke, S.O., Rambau, J.: Online dial-a-ride problems: Minimizing the completion time. In: Proceedings of the 17th International Symposium on Theoretical Aspects of Computer Science, vol. 1770, pp. 639–650. Springer (2000)
2. Atallah, M.J., Kosaraju, S.R.: Efficient solutions to some transportation problems with applications to minimizing robot arm travel. SIAM Journal on Computing **17**, 849–869 (1988)
3. Ausiello, G., Feuerstein, E., Leonardi, S., Stougie, L., Talamo, M.: Competitive algorithms for the traveling salesman. In: Proceedings of the 4th Workshop on Algorithms and Data Structures (WADS'95), *Lecture Notes in Computer Science*, vol. 955, pp. 206–217 (1995)
4. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press (1998)

5. Crites, R.H., Barto, A.G.: Improving elevator performance using reinforcement learning. In: S. Touretsky D. C. Mozer M. E. Hasselmo M. (eds.) Advances in Neural Information Processing Systems 8. MIT Press, Cambridge MA (1996)
6. Crites, R.H., Barto, A.G.: Elevator group control using multiple reinforcement learning agents. Machine Learning **33**(2–3), 235–262 (1998)
7. Fiat, A., Woeginger, G.J. (eds.): Online Algorithms: The State of the Art, *Lecture Notes in Computer Science*, vol. 1442. Springer (1998)
8. Frederickson, G.N., Guan, D.J.: Nonpreemptive ensemble motion planning on a tree. Journal of Algorithms **15**, 29–60 (1993)
9. Frederickson, G.N., Hecht, M.S., Kim, C.: Approximation algorithms for some routing problems. SIAM Journal on Computing **7**, 178–193 (1978)
10. Gross, D., Harris, C.: Fundamentals of queueing theory. Wiley Series in Probability and Statistics. Wiley (1998)
11. Grötschel, M., Hauptmeier, D., Krumke, S.O., Rambau, J.: Simulation studies for the online dial-a-ride-problem. Preprint SC 99-09, Konrad-Zuse-Zentrum für Informationstechnik Berlin (1999). URL http://opus4web.zib.de/documents-zib/401/SC-99-09.pdf. Extended abstract accepted for presentation at Odysseus 2000, first workshop on freight transportation and logistics, Crete, 2000
12. Hauptmeier, D., Krumke, S.O., Rambau, J.: The online dial-a-ride problem under reasonable load. In: Proceedings of the 4th Italian Conference on Algorithms and Complexity, *Lecture Notes in Computer Science*, vol. 1767, pp. 137–149. Springer (2000)
13. Hauptmeier, D., Krumke, S.O., Rambau, J., Wirth, H.C.: Euler is standing in line—dial-a-ride problems with FIFO-precedence-constraints. Discrete Applied Mathematics **113**, 87–107 (2001)
14. Hiller, B., Vredeveld, T.: Probabilistic analysis of online bin coloring algorithms via stochastic comparison. In: D. Halperin, K. Mehlhorn (eds.) ESA, *Lecture Notes in Computer Science*, vol. 5193, pp. 528–539. Springer (2008)
15. Kellerer, H., Tautenhahn, T., Woeginger, G.: Approximability and nonapproximabiblity results for minimizing total flow time on a single machine. In: Proceedings of the Symposium on the Theory of Computing (1996)
16. Klug, T., Hiller, B., Tuchscherer, A.: Improving the performance of elevator systems using exact reoptimization algorithms. Tech. Rep. 09-05, Konrad-Zuse-Zentrum für Informationstechnik Berlin (2009)
17. Krumke, S.O.: Competitive analysis and beyond. Habilitationsschrift, Technische Universität Berlin (2002)
18. Krumke, S.O., Rambau, J., Torres, L.M.: Realtime-dispatching of guided and unguided automobile service units with soft time windows. In: R.H. Möhring, R. Raman (eds.) Algorithms – ESA 2002, 10th Annual European Symposium, Rome, Italy, September 17–21, 2002, Proceedings, *Lecture Notes in Computer Science*, vol. 2461. Springer (2002)
19. Putermann, M.L.: Markov Decision Processes. Wiley Interscience (2005)
20. Rambau, J.: Deferment control for reoptimization – how to find fair reoptimized dispatches. In: S. Albers, R.H. Möhring, G.C. Pflug, R. Schultz (eds.) Algorithms for Optimization with Incomplete Information, no. 05031 in Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum (IBFI), Schloss Dagstuhl, Germany (2005). URL http://drops.dagstuhl.de/opus/volltexte/2005/66 [date of citation: 2010-12-28]
21. Shmoys, D.B., Wein, J., Williamson, D.P.: Scheduling parallel machines on-line. SIAM Journal on Computing **24**(6), 1313–1331 (1995)
22. Tarău, A.: Model-based control for postal automation and baggage handling. Ph.D. thesis, Technische Universiteit Delft (2010)