Konrad-Zuse-Zentrum
für Informationstechnik Berlin

Takustraße 7
D-14195 Berlin-Dahlem
Germany

BENJAMIN HILLER TORSTEN KLUG
ANDREAS TUCHSCHERER

An Exact Reoptimization Algorithm for the Scheduling of Elevator Groups

ZIB-Report 12-43 (November 2012, revised April 2013)

Herausgegeben vom
Konrad-Zuse-Zentrum für Informationstechnik Berlin
Takustraße 7
D-14195 Berlin-Dahlem

Telefon: 030-84185-0
Telefax: 030-84185-125

e-mail: bibliothek@zib.de
URL: http://www.zib.de

ZIB-Report (Print) ISSN 1438-0064
ZIB-Report (Internet) ISSN 2192-7782

An Exact Reoptimization Algorithm for the Scheduling of Elevator Groups*

Benjamin Hiller
Zuse Institute Berlin
Takustraße 7
D–14195 Berlin, Germany.
hiller@zib.de

Torsten Klug
Zuse Institute Berlin
Takustraße 7
D–14195 Berlin, Germany.
klug@zib.de

Andreas Tuchscherer

April 9, 2013

Abstract

The task of an elevator control is to schedule the elevators of a group such that small waiting and travel times for the passengers are obtained. We present an exact reoptimization algorithm for this problem. A reoptimization algorithm computes a new schedule for the elevator group each time a new passenger arrives. Our algorithm uses column generation techniques and is, to the best of our knowledge, the first exact reoptimization algorithm for a group of passenger elevators. To solve the column generation problem, we propose a Branch & Bound method. The overall algorithm finds high-quality solutions very quickly.

1 Introduction

The scheduling of passenger elevators in a building is one of the prime real-world examples of an *online optimization problem*, where the data describing the optimization problem becomes available over time. A suitable elevator scheduling algorithm should make best use of the capacity the elevator group offers to achieve good service, i. e., short waiting times, for the passengers.

Most algorithms in current use are *reoptimization algorithms*: As new information becomes available, they update the schedule to follow by computing a new schedule for the new system state that is supposed to minimize some objective modeling service quality. Industry practitioners often argue (e. g., [18]) that computing an optimal schedule is not possible since the problem is NP-hard and thus justify using heuristics. Recent work shows, however, that reoptimization algorithms based on exact techniques are feasible for similar problems [13, 5].

Most of the currently installed elevator systems are *conventional (2-button) systems*, where a passenger registers his desired travel direction using up/down buttons. The destination floor is entered inside the elevator after it arrived.

*A preliminary extended abstract of this paper was presented at MAPSP 2009 [8].

Thus there is not only uncertainty about future passengers (the online aspect), but also uncertainty about the destination floors of the passengers waiting at a floor. This additional lack of information severely limits the optimization that can be performed. Some elevator companies therefore offer *destination (hall) call systems*, where a passenger registers the destination floor already at his start floor. Such a destination call system provides more information earlier, which should allow to improve the performance of the system.

As far as we know, all existing installations of destination call systems are *immediate assignment (IA) systems*, i. e., passengers are assigned to a serving elevator in immediate response to a call. The immediate assignment limits the scheduling decisions later on and thus reduces the optimization potential. We therefore also consider *delayed assignment (DA) systems*, in which the elevator control can defer the decision which elevator serves a call until some time before the elevator arrives at the floor. It then signals the destination floors served by the elevator, thus selecting the corresponding passengers. DA systems are interesting for two reasons: They might be implemented if they provide better performance and they are a natural yardstick for evaluating algorithms for IA systems.

Contribution We introduce a unified model for both IA and DA systems, allowing to develop algorithms that can handle both, which facilitates system comparisons.

Based on this model, we present the first exact reoptimization algorithm for scheduling a group of passenger elevators in a destination call system. It employs a set partitioning model that is solved via Branch & Price, where the pricing problem is solved by a suitable Branch & Bound method. We report that destination call systems controlled by our new algorithm achieve significantly better waiting times than conventional 2-button systems and a heuristic state-of-the-art destination call algorithm.

The exact approach allows to assess the quality of the solutions of heuristics and is thus an important ingredient for improving real-world scheduling algorithms. The algorithm is a result of a research project with our industry partner Kollmorgen Steuerungstechnik.

Related work Although there is much literature on elevator control algorithms, there is not much work on destination call systems yet. Gloss [6] introduced the idea of destination call systems in 1970, but found that computing power was insufficient to even schedule a single elevator optimally. Interestingly, he in fact proposed DA systems. The first destination call system was introduced around 1990 by Schindler [14]. Seckinger and Koehler [15] reinvestigated the problem for a single elevator and proposed an exact algorithm that frequently obtained optimal solutions in less than a second. Tanaka et al. [17] propose a sophisticated Branch & Bound algorithm for controlling a single elevator, being fast enough for simulations. Both Seckinger and Koehler and Tanaka et al. report that destination call systems achieve lower waiting and travel times than 2-button systems. Conceptually, our algorithm is similar to the vehicle routing algorithm presented in [13], which was later extended to groups of cargo elevators [5] with unit capacity. However, scheduling passenger elevators is much more complex due to their high capacity and the complex constraints involved.

The industry standard is to use relatively simple heuristics [14, 16, 12, 10]. The work presented here is an outgrowth of the PhD thesis [7] of the first author.

The remainder of the paper is structured as follows. Section 2 defines the problem in detail. We describe our new algorithm ExactReplan in Section 3 and provide some computational results on its performance in Section 4.

2 Problem definition

In a destination call system, a passenger registers his destination floor upon his arrival at the start floor. A *destination call* (or *call* for short) is a triple of the release time, the start floor, and the destination floor corresponding to this registration. Note that the elevator control knows only about destination calls, not about passengers. The task of the elevator control is to schedule the elevators such that for each call there is an elevator travelling to the call's start floor to *pick it up* and to visit its destination floor afterwards for *dropping it*, ensuring service of registered passengers. We call a set of schedules for each elevator that serve all calls in this way a *dispatch*. At any point in time, the elevators follow the current dispatch, which is updated each time a new call is registered by solving a *snapshot problem* describing the current system state.

Throughout the paper and in the implementation we assume there is a one-to-one correspondence between destination calls and passengers. Of course, in reality it might (and will) happen that several people register just a single destination call. In this case, we underestimate the number of passengers in the elevator cabin. This might not be an issue, since we do not fill a cabin up to the rated cabin capacity, but only to 80% (as is common practice in the elevator industry), so there is some flexibility. In the worst case, however, some passenger(s) may not enter the cabin as expected and will have to reissue their destination call. This is very similar to the 2-button system, where the information about the number of waiting passengers is even less reliable.

In the following, we describe the structure of the snapshot problem. We do this based on a generic model for elevator schedules that captures 2-button systems, IA systems, and DA systems at the same time. Moreover, the schedule model also applies to (IA/DA) destination call systems with *passenger selection*, where the elevator control can decide and control which passengers board the elevator on each floor. Although this is an unrealistic setting for passenger elevators, it may be interesting as a kind of "yardstick" in performance evaluations.

We will use the following terms for 2-button systems: A *landing call* corresponds to a selected travel direction at a floor (or landing), whereas a *car call* may be given inside the elevator cabin to stop at a certain floor. A *call* may be either a landing call or a destination call.

Request model Due to the way the passengers and the elevator control interact it may not be possible to pick up several calls at the same floor independently. For instance, if in an IA system there are two calls from floor 1 upwards that both have been signalled to be served by elevator 2, then both have to be picked up as elevator 2 arrives and travels upwards. For this reason, we group calls with the same start floor that have to be picked up by the same elevator into *requests*. A request models that the elevator control has no way to distribute its calls among different elevators.

The set of requests $\mathcal{R}$ can be partitioned into the requests that are already assigned to an elevator e, denoted by $\mathcal{R}(e)$, and the ones that are still unassigned, denoted by $\mathcal{R}_u$. The exact set of calls belonging to a request and the sets of assigned requests depend as follows on the system considered.

2-button system: There is one request for each landing call, which consists of this landing call only. A request is assigned to an elevator as soon as the elevator is going to stop at that floor and signalled the corresponding leaving direction.

IA system: There is one request for each floor/direction/elevator combination such that a call from that floor leaving in the direction has been assigned to the elevator. This request consists of all calls with matching floor/direction that have been assigned to the elevator e and belongs to $\mathcal{R}(e)$. Another type of request arises from each yet unassigned call, making up an unassigned request of its own, reflecting that it still can be assigned to any elevator.

DA system: For each elevator e that signalled destination floors to be served, there is a request in $\mathcal{R}(e)$ consisting of all the calls originating from the start floor and going to the destination floors signalled. Unassigned calls belong to the unassigned requests in $\mathcal{R}_u$; there is one such request per start floor/destination floor combination.

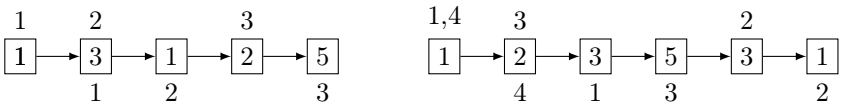
(IA/DA) systems with passenger selection: Each destination call constitutes its own request. The assignment to an elevator is handled as in the system without passenger selection. However, if the capacity does not suffice to pick up all matching requests, the elevator control algorithm may choose the ones to pick up.

A request is served by picking up the corresponding calls and visiting each destination floor of the destination calls belonging to the request. In addition to serving picked up requests, an elevator may have *drop commitments*, i. e., floors to stop at before changing the direction, due to destination calls that have been picked up earlier or due to car calls.

The example mentioned above also shows that if an elevator reaches a floor with an assigned request the first time with leaving direction matching the request direction, it has to pick up this request because the passengers assume the elevator to serve them. We call this the *first-stop pickup requirement*. Observe that the original reason for stopping at this floor might be to drop calls, i. e., there is some drop commitment for this floor. Still the elevator then needs to serve the calls of the first-stop pickup request as well. Thus serving an additional request may have a significant impact on other requests, too. See Figure 1 for an example.

The requests represent the calls still to be picked up. In general, some destination calls have already been picked up, but not been dropped yet. These are represented by the set $\mathcal{C}(e)$ of destination calls loaded by elevator e.

Schedule and feasibility A *schedule* $S = (s_0, \ldots, s_k)$ for elevator e is a sequence of *stops* s_i which describe future visits to floors. A stop s_i is characterized by the stopping floor $s_i.\mathit{floor}$, the leaving direction $s_i.\mathit{direction}$, the arrival



(a) Original schedule S.

(b) Schedule S after including request 4.

Figure 1: Serving request $4\colon 1 \to 2$ changes the structure of the schedule completely. Our graphical notation represents a stop by a boxed floor number. Numbers above or below a stop indicate requests picked up or dropped, respectively.

time $s_i.arrival$, the set of currently loaded calls $s_i.current$, the set $s_i.pickups$ of *requests* picked up at this stop, and the set $s_i.drops$ of *calls* dropped at this floor.

At any point in time, elevator e is either halting at a floor or travelling towards a floor. Let $f_0(e)$ be this floor. Depending on its current state, elevator e may not stop on every floor. For instance, it may currently bypass floor 5 at maximum speed, but deceleration takes too long to stop on floor 6, so floor 7 is the next floor it can stop at. We denote by $\mathcal{F}_i(e)$ the set of floors that are admissible for the first stop in a schedule. If the elevator is halting at floor $f_0(e)$, we have $\mathcal{F}_i(e) = \{f_0(e)\}$ and it is not possible to have the first stop at a floor different from $f_0(e)$. In case the elevator is full, the only floor allowed as a first stop is the next floor of the drop commitments.

Similarly, the leaving direction of the first stop might already be fixed due to drop commitments or because the leaving direction has already been signalled to the passengers. Denote by $D_0(e)$ the set of feasible leaving directions for the first stop. Finally, $\mathcal{C}(e)$ denotes the set of destination calls that are currently loaded by elevator e.

We assume that passengers are well-behaving, i. e., they board the elevator that was signalled to serve them on its first stop in the matching direction. To model this (i. e., the first-stop pickup requirement), we let $P(e, f, d)$ denote the set of requests that are to be picked up by elevator e at its first stop on floor f with leaving direction d. The first such stop of elevator e then has to be followed by stops according to the additional drop commitments implied by all requests in $P(e, f, d)$. The definition of $P(e, f, d)$ depends as follows on the considered system.

2-button system: All sets $P(e, f, d)$ are empty. The rationale for this is that in a conventional system there are no implied drop commitments, since the elevator control does not know the destination floors yet.

IA system: $P(e, f, d)$ is the set of requests leaving floor f that have been assigned to elevator e.

DA system: $P(e, f, d)$ is again the set of requests leaving floor f that have been assigned to elevator e. Note that in a delayed assignment system, requests are only assigned to an elevator that is already approaching the floor and signalled the corresponding destination floors.

(IA/DA) systems with passenger selection: All sets $P(e, f, d)$ are empty,

since the elevator can freely choose which passengers/requests are picked up.

We now have all the ingredients to formally define feasibility of a schedule $S = (s_0, \ldots, s_k)$ for elevator e. For convenience, we denote by $P(s_i)$ the set of requests already picked up by the schedule up to, but not including, stop s_i. Note that $P(s_0) = \emptyset$.

The most important feasibility restrictions are due to the requirement that (well-behaving) passengers must not be transported in the wrong direction. This implies restrictions on the sequence of floors and the leaving directions of the stops. For the initial stop s_0 we have the following conditions:

- The floor $s_0.\mathit{floor}$ is one of the floors of $\mathcal{F}_i(e)$.
- The leaving direction $s_0.\mathit{direction}$ is one of $D_0(e)$.
- The drop commitments $s_0.\mathit{drop\_floors}$ are the floors from $\mathcal{F}(e)$ without $s_0.\mathit{floor}$ plus the destination floors of the destination calls picked up at this stop.

The analogues of these conditions for the later stops are a bit more involved.

- There are the following cases depending on the pending drop commitments:

  $s_i.\mathit{drop\_floors} \neq \emptyset$**:** $s_{i+1}.\mathit{floor}$ is between $s_i.\mathit{floor}$ and the next floor from $s_i.\mathit{drop\_floors}$ in the leaving direction of s_i or equal to this floor.
  The leaving direction $s_{i+1}.\mathit{direction}$ is the same as on the last stop, if $s_i.\mathit{drop\_floors}$ consists of at least two floors or it is one floor, but $s_{i+1}.\mathit{floor}$ is before this floor. Otherwise the direction may be arbitrary.

  $s_i.\mathit{drop\_floors} = \emptyset$**:** $s_{i+1}.\mathit{floor}$ and the leaving direction are arbitrary.

- The drop commitments $s_{i+1}.\mathit{drop\_floors}$ are the same as $s_i.\mathit{drop\_floors}$ without $s_{i+1}.\mathit{floor}$ plus the destination floors of the destination calls picked up at this stop.

In addition to the conditions due to passenger directions there are conditions on the sets of picked up requests and the sets of current and dropped destination calls.

- $s_i.\mathit{pickups}$ contains all the requests in $P(e, s_i.\mathit{floor}, s_i.\mathit{direction}) \setminus P(s_i)$. Moreover, $s_i.\mathit{pickups}$ does not contain any request that is assigned to a different elevator and, of course, each request starts at floor $s_i.\mathit{floor}$ and travels in direction $s_i.\mathit{direction}$.
- $s_i.\mathit{current}$ is the set of current calls of the preceding stop ($\mathcal{C}(e)$ for s_0) that have a different destination floor than $s_i.\mathit{floor}$ plus the destination calls corresponding to $s_i.\mathit{pickups}$. Note that these are all calls that could have been picked up if the elevator capacity was not limited.
- $s_i.\mathit{drops}$ is the set of current calls of the preceding stop ($\mathcal{C}(e)$ for s_0) that have destination floor $s_i.\mathit{floor}$.

Now the overall schedule is feasible if and only if

- every stop is feasible,
- there are no two successive stops on the same floor with the same leaving direction,
- all requests assigned to the elevator are served, i. e., $\mathcal{R}(e) \subseteq P(s_k)$,
- finally, all calls and drop commitments have been served, i. e., $s_k.\mathit{current} = \emptyset$ and $s_k.\mathit{drop\_floors} = \emptyset$.

Timing and cost model We use the following timing model for schedules. The floor-to-floor travel time from floor f_1 to floor f_2 of elevator e is given by $\tau_{\mathrm{drv}}(e, f_1, f_2)$ and includes the time to close and reopen the door. Moreover, let τ_{stop} be the minimum stopping time on a floor and τ_{load} the transfer time of a passenger, i. e., the time a passenger needs for boarding or leaving. Given the information of a stop s_i, the arrival time of the subsequent stop s_{i+1} is given by

$$\begin{aligned} s_{i+1}.\mathit{arrival} = s_i.\mathit{arrival} &+ \max\{\tau_{\mathrm{stop}}, \tau_{\mathrm{load}}\big(|s_i.\mathit{pickups}| + |s_i.\mathit{drops}|\big)\} \\ &+ \tau_{\mathrm{drv}}(e, s_i.\mathit{floor}, s_{i+1}.\mathit{floor}). \end{aligned} \tag{1}$$

Note that the elevator stopping time at a floor is essentially proportional to the number of calls transfered.

Each feasible schedule is associated to a cost, which is the (weighted) sum of the waiting and travel times. The *waiting time* is the time between the arrival of a passenger and the time the serving elevator arrives. Similarly, the *travel time* is the time between the arrival of the passenger and the arrival of the serving elevator at the destination floor. Based on the arrival times it is easy to compute the waiting times for the calls picked up according to $s_i.\mathit{pickups}$ and the travel times for the calls in $s_i.\mathit{drops}$. Let $t_{\mathrm{wait}}(c)$ and $t_{\mathrm{travel}}(c)$ denote the waiting and travel time of call c, respectively. The cost for serving call c is then $c_{\mathrm{wait}} t_{\mathrm{wait}}(c) + c_{\mathrm{travel}} t_{\mathrm{travel}}(c)$ where c_{wait} and c_{travel} are preselected cost coefficients.

So far we did not take the limited elevator capacity into account. In general, we cannot avoid insufficient elevator capacity so we cannot strictly forbid capacity violations. To avoid overbooking and thus having the passengers to reissue their calls, we introduce a capacity penalty cost as follows. We penalize each call of a request that exceeds the elevator capacity *at the pickup stop of the request* with additional cost c_{capacity}. Denoting the capacity of elevator e by $\kappa(e)$, the capacity penalty cost at stop s are

$$c_{\mathrm{capacity}} \cdot \max\big\{0, |s.\mathit{current}| + |s.\mathit{pickups}| - \kappa(e)\big\}.$$

A *dispatch* is a collection of feasible schedules, one for each elevator, that serves every request exactly once. The cost of a dispatch is just the sum of costs of the schedules. The overall task when solving the snapshot problem is to determine a dispatch with minimal cost. Note that a dispatch determines an assignment of requests to elevators, which in an IA system determines the assignment of calls to elevators for the next snapshot problem.

3 The algorithm

Our algorithm ExactReplan is designed to work for destination call systems (and not for 2-button systems), since it assumes that the assignment of one request to an elevator is independent of the schedules of the remaining elevators. This is not the case for 2-button systems, since here essentially the elevator to arrive first will serve a landing call. We also note that although the algorithm is in principle applicable to systems with passenger selections, the resulting instances will be to huge to be solved in a reasonable time, since each passenger constitutes a request on its own. In contrast, for IA and DA systems without passenger selection, 100 or more destination calls are usually grouped into not more than 30 requests.

The algorithm ExactReplan is based on the following set partitioning model. Recall that a dispatch distributes the unassigned requests $\mathcal{R}_u$ among the elevators and gives, for each elevator, a feasible schedule that serves its assigned requests $\mathcal{R}(e)$ and the unassigned requests it received.

Let E denote the set of elevators. Furthermore, let $\mathcal{S}(e)$ be the set of feasible schedules for elevator e and define $\mathcal{S} := \bigcup_{e \in E} \mathcal{S}(e)$. A dispatch is then a collection of feasible schedules such that each *unassigned* request is served exactly once. For each schedule $S \in \mathcal{S}$ we introduce a decision variable $x_S \in \{0, 1\}$ for including it in the dispatch or not. We denote the cost of schedule S by $c(S)$. To compute an optimal dispatch, we can solve the following set partitioning problem:

$$\min \sum_{S \in \mathcal{S}} c(S) x_S \tag{2}$$

$$\sum_{S \in \mathcal{S}:\ \rho \in S} x_S = 1 \qquad \rho \in \mathcal{R}_u, \tag{3}$$

$$\sum_{S \in \mathcal{S}(e)} x_S = 1 \qquad e \in E, \tag{4}$$

$$x_S \in \{0, 1\} \quad S \in \mathcal{S}. \tag{5}$$

This Integer Programming (IP) problem cannot be solved by an off-the-shelf IP solver directly since the number of feasible schedules and thus columns is very large. It is, however, sufficient to consider for each elevator e only cost-minimal feasible schedules for each subset $R \subseteq \mathcal{R}_u$, which are just a few if $|\mathcal{R}_u|$ is not large, as it is usually the case for IA systems. It turns out that generating only the cost-minimal feasible schedules using the Branch & Bound algorithm described below takes more than a second already for 3 or 4 unassigned requests for a group of 8 elevators. Our algorithm therefore uses column generation (see e. g., [4]) to solve the linear programming relaxations of the above IP arising in a Branch & Price process. We do not invoke the column generation/IP solver if there is only a single unassigned request. Instead, we determine the optimal dispatch by computing an optimal schedule serving this request for each elevator and then choosing the elevator / schedule with least additional cost. Having only one unassigned request is very frequent for IA systems.

Let π_ρ, $\rho \in \mathcal{R}_u$, and π_e, $e \in E$, denote the dual prices associated with constraints (3) and (4), respectively. The pricing problem for the LP relaxation of the above IP is then to find, for each elevator e, a set of feasible schedules for

elevator e having negative reduced cost $\tilde{c}(S) := c(S) - \sum_{\rho \in \mathcal{R}_u \cap S} \pi_\rho - \pi_e$ or to decide that no such schedule exists.

3.1 Pricing via Branch & Bound

Pricing problems for similar vehicle routing problems can often be modelled as shortest path problems. Due to the complex constraints of the elevator scheduling problem, the corresponding graph is very large. We therefore solve the pricing problem using a Branch & Bound algorithm which allows us to take care of all the constraints rather easily. Another important feature of our pricing algorithm is that we do not solve the pricing problem exactly, i. e., determine the schedule with the smallest (negative) reduced cost, but stop pricing after we found k schedules with negative reduced cost. These schedules are then added to the set partitioning IP. The rationale is that we do not spend too much search effort initially, when the dual prices are still far from the optimal ones and thus do not yield reliable valuations of the relative usefulness of schedules.

The basic observation to enumerate all feasible schedules for an elevator is that each schedule can be described by a sequence of pickup actions and drop actions, specifying which request to pick up and at which floor to stop for dropping, respectively. We use this description to make up a search tree traversed to find schedules with negative reduced costs. Each node in the tree corresponds to a stop of the elevator at a floor, where a subset of the possible requests has already been picked up. The nodes are joined by edges corresponding to pickup or drop actions, if the action does not lead to transporting a passenger in the wrong direction.

Formally, a node v of the search tree is labelled with the following data:

A_v Set of not yet picked up (assigned) requests from $\mathcal{R}(e)$.

O_v Set of not yet picked up (optional) requests from $\mathcal{R}_u$.

S_v A feasible schedule that serves all requests in $\mathcal{R}(e) \setminus A_v$. This schedule is determined by the path from the root node to node v. In addition to stops corresponding to the actions on this path S_v has stops at the pending drop floors.

s_v A stop from S_v, namely the stop resulting from the sequence of actions along the path from the root node.

Node v represents all schedules that may be obtained by picking up unserved requests at stop s_v or later. The schedule S_v has the property that there are no pickups after stop s_v, i. e., these stops are there only for dropping loaded destination calls. Node v is called *feasible* if A_v is empty and s_v is the last stop of S_v, i. e., has no pending drop floors. A feasible node corresponds to a feasible schedule that serves all requests assigned to the elevator. There is a separate root node r for every floor f where the elevator can still stop at next. S_r is the schedule corresponding to dropping all loaded calls of elevator e with first stop at floor f; s_r is the first stop of S_r.

As described before, a child node v' of node v arises by two actions: Either a request is picked up or the elevator moves to the next floor for dropping a loaded call. The child node v' for dropping a call has the labels of v with $s_{v'}$ being the successor stop of s_v. If v' reflects the pickup of request ρ, it is

labeled with schedule $S_{v'}$ arising from S_v in the following way. The stops up to s_v and all stops that need to be visited before ρ can be picked up are copied directly. If necessary, a stop at the start floor of ρ is created, potentially also including pickups according to the first-stop pickup requirement. Then stops for dropping the now loaded calls are appended. Of course, $A_{v'}$, $O_{v'}$, and $s_{v'}$ are set accordingly.

Our Branch & Bound pricing algorithm starts by computing lower bounds on the reduced cost for each root node and collects the root nodes in a queue Q. It traverses the search forest in a best-first manner, i. e., it selects the node v from Q with the smallest lower bound. Moreover, the algorithm maintains a result set M, containing all schedules with negative reduced cost found so far. Each node v is processed as follows: If v is feasible and has reduced cost less than threshold θ (initially -1.0×10^6), the corresponding schedule is collected in the result set M. If M has now k schedules, the search is stopped and M returned to be added to the set partitioning IP. Otherwise we branch on v and collect all valid child nodes in the set N. Each $u \in N$ is added to Q provided that its lower bound is less than θ. Then we consider the next node from Q. Everytime a schedule is added to M, θ is set to the minimum reduced cost of a schedule in M to avoid finding many schedules whose reduced cost are only slightly negative. Thus we can prune a large part of the search tree, looking only for schedules with even lower negative reduced cost.

3.2 Lower bounds

The lower bound computed consists of two parts: a lower bound on the reduced cost of the requests already picked up and dropped or currently loaded and a lower bound on the *additional* reduced cost for serving still unserved requests. The reduced cost for the picked up requests are at least $\tilde{c}(S_v)$. To see this, note that the service costs increase with time. Due to the construction of S_v, the waiting times of all calls are already fixed and the travel time of currently loaded calls can only increase by picking up further requests, since the calls cannot be dropped earlier than at the already existing drop stops. Since capacity penalty cost are only incurred when picking up a request and all pickups of S_v are irrevocable, they cannot decrease either.

We now turn to the computation of the lower bound for the *additional* reduced cost. The idea of this "greedy"-type bound is to determine for each unserved request earliest pickup and drop times. Serving the request incurs at least the service cost according to these times, so the sum of all of these costs gives a lower bound for the additional cost. Note that this corresponds to assuming that *all* requests can be served at their earliest pickup times, i. e., the interactions between requests due to serving a set of them are neglected.

Seckinger and Koehler [15] used this idea in a Branch & Bound algorithm to compute an optimal schedule for a given set of requests. However, they did not take into account the currently loaded calls and the requirement that passengers must not travel in the opposite direction as we do. Thus our lower bound is significantly stronger.

Consider a node v in the search tree and a request $\rho \in A_v \cup O_v$ and let $f^+(\rho)$ be the start floor of ρ. In case the direction of ρ is opposite to the leaving direction from s_v, the elevator has to visit all drop floors before it can pickup ρ. If the direction matches, the elevator has to pass the drop floors before (and

including) $f^+(\rho)$, too. For both cases, let s' be the first stop of S_v where all preceding drop floors have been passed. A lower bound $\underline{t}^+(\rho)$ for the pickup time of ρ is then given by

$$\underline{t}^+(\rho) = \begin{cases} s'.\mathit{arrival} & f^+(\rho) = s'.\mathit{floor}, \\ s'.\mathit{arrival} + \tau_{\text{stop}} + \tau_{\text{drv}}(e, s'.\mathit{floor}, f^+(\rho)) & f^+(\rho) \neq s'.\mathit{floor}. \end{cases}$$

Now consider a call $c \in \rho$ and let $f_1, \ldots, f_\ell$ be the sequence of ℓ floors to be visited before dropping c, where $f_1 = f^+(\rho)$ and f_ℓ is the destination floor of call c. This sequence includes the drop floors of the currently loaded calls of stop s' and the destination floors of other calls of ρ. A lower bound $\underline{t}^-(c)$ for the drop time of c is

$$\underline{t}^-(c) = \underline{t}^+(\rho) + \max\{\tau_{\text{stop}}, |\rho|\tau_{\text{load}}\} + \sum_{i=1}^{\ell-1} \tau_{\text{drv}}(e, f_i, f_{i+1}) + (\ell - 2)\tau_{\text{stop}}.$$

To estimate the additional capacity penalty cost at node v or below we have to treat the immediate assignment system and the delayed assignment system differently, due to the differences in signalling. In general, there are two types of contributions to the additional capacity penalty cost: costs $\underline{c}_{\text{cap}}(\rho)$ that can be attributed to picking up a request ρ, and costs $\underline{c}_{\text{cap}}(S_v, A_v)$ that arise from A_v and the structure of S_v so far. Intuitively, the latter part is either due to assigned requests at a floor / direction pair exceeding the cabin capacity, or due to earlier picking up requests that now take up too much capacity to accomodate the remaining assigned requests.

Recall that in the immediate assignment system, a request has to be picked up on the first stop with matching start floor and leaving direction. Denote by $n(f, d)$ the number of calls that are unavoidably picked up when leaving floor f in direction d. The value of $n(f, d)$ is given by the number of all *assigned* requests (i. e., requests in A_v) starting at floor f with travel direction d. In addition, let $\kappa(f, d)$ be the remaining cabin capacity when the elevator leaves floor f in direction d after a subsequent stop. More precisely, we have $\kappa(f, d) = \max\{0, \kappa(e) - |s.\mathit{current}|\}$, if there is a stop s on floor f with leaving direction d in schedule S_v equal to or after the current stop s_v. If there is no such stop, $\kappa(f, d) = \kappa(e)$. We can now bound $\underline{c}_{\text{cap}}(S_v, A_v)$ by

$$\underline{c}_{\text{cap}}(S_v, A_v) = c_{\text{capacity}} \cdot \left(\sum_{f,d} \max\{0, n(f, d) - \kappa(f, d)\} \right).$$

Since the assigned requests are also accounted for by $\underline{c}_{\text{cap}}(S_v, A_v)$, we set $\underline{c}_{\text{cap}}(\rho) = 0$ for all $\rho \in A_v$. For optional requests, the additional capacity penalty costs can only be estimated for each request separately. However, it is possible to take into account the assigned requests. To this end, let $n(\rho) := n(f, d)$ and $\kappa(\rho) = \kappa(f, d)$ for the start floor f and direction d of request ρ. For $\rho \in O_v$ we then have the bound

$$\underline{c}_{\text{cap}}(\rho) = \begin{cases} c_{\text{capacity}} \cdot \big(\max\{0, |\rho| + n(\rho) - \kappa(\rho)\}\big) & n(\rho) < \kappa(\rho), \\ c_{\text{capacity}} \cdot |\rho| & n(\rho) \geq \kappa(\rho). \end{cases}$$

In the first case, there is some cabin capacity left that may even accomodate the loaded calls and those from ρ. In the second case, no cabin capacity is left so every call of ρ incurs the capacity penalty.

For the delayed assignment system, we use $\underline{c}_{\text{cap}}(S_v, A_v) = 0$. The rationale is that exceeding the capacity due to too many loaded calls can be avoided by first emptying the elevator, i. e., signalling no destination floor at the remaining drop floors of S_v. Since every request can effectively be served on its own the only way to incur a capacity penalty is that a single request already exceeds the elevator capacity, i. e., $|\rho| > \kappa(e)$. This leads to the bound $\underline{c}_{\text{cap}}(\rho) = c_{\text{capacity}} \cdot \max\{0, |\rho| - \kappa(e)\}$ for each $\rho \in A_v \cup O_v$.

Denoting the release time of a call c by t_{release} (c), we can use these estimates to obtain a lower bound $\underline{c}(\rho)$ on the additional cost for serving request ρ in any schedule by summing the cost according to the time bounds and the capacity penalty cost, i. e.,

$$\underline{c}(\rho) = \sum_{c \in \rho} \Big(c_{\text{wait}}(c)[\underline{t}^+(\rho) - t_{\text{release}}(c)] + c_{\text{travel}}(c)[\underline{t}^-(c) - t_{\text{release}}(c)] \Big) + \underline{c}_{\text{cap}}(\rho).$$

An important observation is that we can do some kind of *dual fixing* based on $\underline{c}(\rho)$ for requests $\rho \in O_v$: If $\pi_\rho \leq \underline{c}(\rho)$ it will never be favorable to serve this request, since it cannot decrease the reduced cost and it does not have to be served by this elevator. These requests can thus be ignored in the subtree below node v, thus pruning the search tree.

3.3 Incorporation in a Branch & Price framework

The ingredients described in the preceding subsections are already sufficient to obtain an algorithm that finds high-quality (though not necessarily optimal) solutions as follows. We solve the LP relaxation of (2)–(5) using the Branch & Bound pricing algorithm. To find a schedule, we then use an IP solver to find the optimal solution of IP (2)–(5) with the restricted set of schedules encountered during column generation.

To obtain a truely exact algorithm that always finds an optimal solution we need to incorporate the column generation into a Branch & Price framework. In Branch & Price, column generation is applied at all nodes generated during the usual LP-based Branch & Bound search to solve the LP relaxations. It is crucial that the branching is done in such a way that it is compatible with the pricing problem, i. e., the branching decision can be taken into account when solving it.

In our algorithm ExactReplan, we use the following branching strategy. Let x be the optimal fractional LP solution at some node v in the IP search tree. For each unassigned request $\rho \in \mathcal{R}_u$, we consider the set of elevators $E_\rho(x)$ that are used to serve ρ in x, i. e.,

$$E_\rho(x) := \{e \in E \mid \exists S \in \tilde{\mathcal{S}}(e) \colon x_s > 0\},$$

where $\tilde{\mathcal{S}}(e)$ denotes the subset of the schedules generated at node v belonging to elevator e. Since x is fractional, there is at least one $\rho \in \mathcal{R}_u$ with $|E_\rho(x)| \geq 2$. We choose any such ρ and partition $E_\rho(x)$ into two subsets E_1 and E_2 of (almost) the same size such that the condition

$$0 < \sum_{e \in E_i, S \in \tilde{\mathcal{S}}(e) \colon \rho \in S} x_S < 1$$

is satisfied for $i \in \{1, 2\}$. We then create two child nodes of v with the additional constraints

$$\sum_{e \in E_1, S \in \mathcal{S}(e):\ \rho \in S} x_S = 1, \qquad \sum_{e \in E \setminus E_1, S \in \mathcal{S}(e):\ \rho \in S} x_S = 0, \tag{6}$$

and

$$\sum_{e \in E_1, S \in \mathcal{S}(e):\ \rho \in S} x_S = 0, \qquad \sum_{e \in E \setminus E_1, S \in \mathcal{S}(e):\ \rho \in S} x_S = 1, \tag{7}$$

respectively. Clearly, x is cutoff by both (6) and (7). Moreover, it is easy to take care of (6) and (7) in the pricing algorithm: We simply forbid serving ρ for the elevators in $E \setminus E_1$ (or E_1) and force assignment of ρ to elevator e in case $E_1 = \{e\}$ (or $E \setminus E_1 = \{e\}$).

To speed up the pricing process, we use an additional optional technique which we call *pricing of old schedules*. If *pricing of old schedules* is enabled, we keep all schedules from the previous reoptimization run and check whether any of those has negative reduced cost before invoking the Branch & Bound pricing. If schedules with negative reduced cost are found, they are added to the master problem and the current LP is resolved before pricing continues. This technique allows to exploit the fact that two subsequent snapshot problems are rather similar by "warmstarting" the pricing process. If *pricing of old schedules* is used, we get the same LP relaxation value, but usually fewer columns.

4 Computational results

We implemented our algorithms and the simulation environment in C++, using the SCIP 2.0 framework [2, 1] to implement the ExactReplan algorithm, employing CPLEX 12.4 as LP solver. All computations ran under Linux on a system with an Intel i7 870 CPU and 16 GB of RAM. In all the tests performed we used the service cost coefficients of the waiting time set to 1 and 0 for the travel time. The reason is that we want to compare the resulting waiting times to those achieved by a conventional 2-button system in which only the waiting time can be measured. Moreover, the waiting time is the most important criterion for the service quality of an elevator system. The capacity penalty c_{capacity} is chosen such that it corresponds to the waiting time cost of 300 seconds.

Elevator control algorithms are usually evaluated for several kinds of traffic patterns that arise in a typical office building. In the morning, passengers enter the building from the ground floor, causing *up peak traffic*. Then there is some *interfloor traffic* where the passengers travel roughly evenly between the floors. During *lunch traffic*, people leave and reenter the building via the ground floor. Finally, there is *down peak traffic* when people leave the building in the afternoon. In addition, we also consider *real up peak traffic* and *real down peak traffic*, which mix the up peak and down peak traffic with 5% of interfloor and 5% of down peak and up peak traffic, respectively. These two patterns are supposed to model the real traffic conditions more closely than the pure ones.

To evaluate the performance of ExactReplan, we generated 10 snapshots for each of the six traffic patterns. This was done by simulating five minutes of traffic in a 23-floor building ("building B" in Table 4 on page 17). In these five

traffic pattern	calls			requests			unassigned requests		
	min	avg	max	min	avg	max	min	avg	max
IA, Interfloor	35	42.1	48	15	17.0	19	1	1	1
IA, Real Down Peak	52	59.0	70	14	15.6	19	1	1	1
IA, Down Peak	51	64.4	80	13	16.2	21	1	1	1
IA, Lunch Peak	63	69.1	78	15	17.4	19	1	1	1
IA, Real Up Peak	58	76.6	91	6	10.5	14	1	1	1
IA, Up Peak	55	65.3	79	2	3.6	5	1	1	1
DA, Interfloor	29	36.3	42	14	16.4	19	14	16.4	19
DA, Real Down Peak	24	46.6	57	13	15.1	19	13	15.1	19
DA, Down Peak	36	51.4	70	13	14.5	18	13	14.5	18
DA, Lunch Peak	50	63.9	77	16	19.7	28	16	19.7	28
DA, Real Up Peak	63	71.8	79	17	22.4	28	17	22.4	28
DA, Up Peak	30	45.3	59	11	13.7	18	11	13.7	18

Table 1: Overview of the size of the snapshot problems in our test set.

minutes, 16% of the building population had to be transported, which represents a very high load situation and thus a stress test for our algorithm. We used these 16% for all traffic patterns except for interfloor traffic, where we assumed 10% since interfloor traffic usually has a lower intensity. From the recorded snapshot problems of the simulation runs, we selected 10 snapshot problems with long running times. These snapshot problems thus do not reflect typical performance, but rather point to performance limitations of the algorithm and are thus suitable for investigating which techniques may improve the performance. Table 1 summarizes some statistics of our test set. Note that in each of the intermediate assignment snapshot problems there is only one unassigned request. This is an artifact of our simulation, which invokes a reoptimization after each new call. In a real IA system, most often there will be just one new call, but never more than say, 5 new calls.

The performance of ExactReplan on the selected snaphot problems is shown in Table 2. Since there is always just one assigned request in the IA snaphot problems, they are solved by enumeration. This is very efficient and the computation time of ExactReplan is never more than 0.01 seconds in this case.

The DA system snapshots are more challenging, since there are more unassigned requests and we thus need to solve our IP model. Table 2 shows how the optional feature *pricing of old schedules* influences the performance of ExactReplan. As Table 2 reveals, the algorithm in its default variant without pricing of old schedules is already very fast for most situations. It solves all delayed assignment (DA) snapshots from interfloor and down peak traffic in well below a second. *Pricing of old schedules* roughly halves the average running time with the exception of Real Up Peak, where the average running time increases by 5%. A closer look reveals that this is entirely due to the snapshot instance with the maximum running time; ignoring this instance the average running time is reduced by 14% (this data is indicated by an asterisk in Table 2). Up peak and real up peak are then the only traffic situations which require considerably more than one second to solve to optimality. Although the running time for real up peak traffic is up to 24,000 seconds for very high load snapshots, we can

scenario	nodes		time [s]	
	avg	max	avg	max
immediate assignment, default settings				
IA, Interfloor	192	262	0.01	0.01
IA, Down Peak	263	509	0.01	0.01
IA, Real Down Peak	276	524	0.01	0.01
IA, Lunch Peak	280	600	0.01	0.01
IA, Up Peak	35	41	0.00	0.01
IA, Real Up Peak	86	135	0.01	0.01
delayed assignment, default settings				
DA, Interfloor	8,929	31,491	0.18	0.63
DA, Down Peak	7,930	18,347	0.14	0.33
DA, Real Down Peak	8,216	19,962	0.14	0.35
DA, Lunch Peak	46,644	180,335	1.00	4.17
DA, Up Peak	1,259,773	7,008,815	63.96	388.91
DA, Real Up Peak	78,425,653	488,514,111	3,955.88	23,925.48
DA, Real Up Peak*	32,860,269	153,835,629	1,737.03	8,365.17
delayed assignment, pricing of old schedules				
DA, Interfloor	5,013	20,527	0.10	0.41
DA, Down Peak	4,243	11,197	0.07	0.20
DA, Real Down Peak	4,351	15,087	0.08	0.31
DA, Lunch Peak	23,249	97,758	0.51	2.14
DA, Up Peak	590,975	2,526,717	29.13	138.07
DA, Real Up Peak	82,712,301	571,786,012	4,161.57	28,194.00
DA, Real Up Peak*	28,370,778	129,508,242	1,491.30	6,986.47

Table 2: Performance of our exact reoptimization algorithm ExactReplan on the snapshot problem test set. Shown are the total number of nodes in all pricing search trees and the running time. In the lines marked by an asterisk, the outlier instance with the maximum running time is omitted.

conclude that ExactReplan is fast enough to simulate a DA system, allowing us to investigate the additional performance gain that might be achieved by using a DA system instead of an IA system. We also remark that the running time on a DA system rises above one second only for snapshot problems with at least 10 unassigned requests. We conclude from this that ExactReplan has running times less than a second on an IA system, even if there is more than one new call.

Table 3 shows data for the optimality gaps of ExactReplan with and without *pricing of old schedules* after one second running time. An optimal solution for determining the optimality gap has been computed by solving each snapshot problem to optimality using ExactReplan without a time limit. The numbers $\beta_{0.25}$, $\beta_{0.5}$, $\beta_{0.75}$, and β_1 are the 0.25-, 0.5-, 0.75-, 1.0-quantiles of the optimality gap. As we can see, the gaps are almost always less than 1% and *pricing of old schedules* helps to reduce the remaining gap. For comparison, Table 3 also shows the optimality gaps achieved by our heuristic destination call algorithm BestInsert that we developed earlier [10, 7]. BestInsert also obtains solutions with very small optimality gaps, however, ExactReplan is

scenario	$\beta_{0.25}$	$\beta_{0.5}$	$\beta_{0.75}$	β_1	$\emptyset$
BestInsert					
DA, Interfloor	0.00%	0.00%	0.09%	1.28%	0.19%
DA, Down Peak	0.00%	0.00%	0.24%	0.32%	0.08%
DA, Real Down Peak	0.00%	0.00%	0.19%	1.23%	0.21%
DA, Lunch Peak	0.00%	0.00%	0.25%	0.78%	0.14%
DA, Up Peak	0.00%	0.01%	0.22%	0.62%	0.14%
DA, Real Up Peak	0.16%	0.57%	1.02%	2.09%	0.75%
ExactReplan *stopped after one second*					
DA, Interfloor	0.00%	0.00%	0.00%	0.09%	0.01%
DA, Down Peak	0.00%	0.00%	0.00%	0.00%	0.00%
DA, Real Down Peak	0.00%	0.00%	0.00%	0.13%	0.01%
DA, Lunch Peak	0.00%	0.00%	0.00%	0.03%	0.00%
DA, Up Peak	0.00%	0.01%	0.06%	0.62%	0.09%
DA, Real Up Peak	0.16%	0.57%	1.02%	2.09%	0.75%
ExactReplan *stopped after one second, using* pricing of old schedules					
DA, Interfloor	0.00%	0.00%	0.00%	0.09%	0.01%
DA, Down Peak	0.00%	0.00%	0.00%	0.00%	0.00%
DA, Real Down Peak	0.00%	0.00%	0.00%	0.04%	0.00%
DA, Lunch Peak	0.00%	0.00%	0.00%	0.02%	0.00%
DA, Up Peak	0.00%	0.00%	0.04%	0.45%	0.06%
DA, Real Up Peak	0.11%	0.50%	0.60%	2.09%	0.56%

Table 3: Quality of ExactReplan solutions after one second running time.

often able to improve non-optimal solutions of BestInsert within one second running time.

Our results indicate that ExactReplan with *pricing of old schedules* gives the best results after one second. We will therefore use this setting in our simulation experiments.

To study the online performance of our algorithm we consider up peak traffic only, since it is well known [3] that it is the most demanding traffic situation. In order to assess the performance of an elevator scheduling algorithm, we simulate 5 minutes of up peak traffic, recording the waiting times achieved by the algorithm. This is intended to mimic the peak traffic condition in a building. We do this with increasing traffic intensity (measured as a percentage of the building population) and are interested in the highest intensity where the algorithm still provides reasonable service. According to Barney [3], performance is still fair, if the median $\alpha_{0.5}$ of the waiting time is at most 25 seconds and the 0.9-quantile $\alpha_{0.9}$ is at most 55 seconds. We define the *handling capacity* of an elevator system to be the highest traffic intensity, measured as percentage of the overall population that may arrive in 5 minutes, that can be served without exceeding these thresholds.

In our simulations, we limit the computation time of ExactReplan to one second. A new snapshot is computed each time a new call arrives. We

	building A	building B
population	3300	1400
floors	12	23
elevators	8	6
cabin capacity	19	13
acceleration [m/s^2]	1.0	0.9
maximum speed [m/s]	2.0	5.0
deceleration [m/s^2]	1.0	0.9

Table 4: Details of the elevator systems in the buildings considered.

consider two example buildings / elevator systems whose data is given in Table 4. Building A has 12 floors served by a group of eight elevators. Its population is 3,300 persons, which is relatively high. In contrast, the 23-floor building B has only a population of 1,400 persons. The elevator group in building B consists of six elevators with a significantly higher maximum speed than that of the elevators in building A. This high maximum speed is necessary to provide a reasonable service in such a tall building. We are interested in how much the handling capacity of both elevator systems can be improved by applying the advanced destination control algorithms developed in the preceding section.

Before looking at the handling capacity that can be achieved by ExactReplan, we compare ExactReplan to our heuristic destination call algorithm BestInsert [10, 7], which may also be used for both IA and DA systems. For an IA system, the waiting times are almost identical, since there is always only one unassigned request and the schedules therefore do not differ much. Tables 5 and 6 give the waiting times obtained for a DA system in building A and B, respectively. ExactReplan performs as least as good as BestInsert in all cases and outperforms BestInsert considerably for higher load, sometimes even halving both the median $\alpha_{0.5}$ and the 0.9-quantile $\alpha_{0.9}$ of the waiting time. We saw in Table 3 that the relative improvement of ExactReplan over BestInsert in terms of snapshot optimality is rather small, still the solutions found by ExactReplan are much better in the long run. This observation is in line with our findings in an earlier similar application [9].

Finally, we compare the performance of ExactReplan controlling an IA and a DA system with the performance of a conventional 2-button system which is controlled by the CGC algorithm [3, 7]. The CGC algorithm inserts requests (i. e., up/down calls) successively in a set of elevator schedules serving all registered stopping floors plus the requests assigned so far. The cost of request ρ incurred by assigning it to elevator e is the waiting time of ρ in the resulting single-roundtrip schedule for e. In order to compute reasonable estimates for this, CGC assumes that the (unknown) destination floor of each request is halfway between the start floor and the last floor in the request direction. In general, request ρ is assigned to the elevator with minimal cost, with two exceptions based on a parameter HTT called "high threshold time". In order to reduce the number of stops, request ρ is allocated to an elevator with a car call at the start floor of ρ, provided it does not get waiting time greater than HTT due to this allocation. Moreover, request ρ is not allocated to the minimum cost elevator if the waiting time of another request increases beyond HTT. If

Szenario	BestInsert DA system			ExactReplan DA system		
	$\alpha_{0.5}$	$\alpha_{0.9}$	$\emptyset$	$\alpha_{0.5}$	$\alpha_{0.9}$	$\emptyset$
Up 6%	**0**	2	**1**	**0**	**1**	**1**
Up 7%	**0**	**10**	**2**	**0**	**10**	**2**
Up 8%	**0**	**20**	**5**	**0**	**20**	**5**
Up 9%	**2**	33	**11**	3	**32**	**11**
Up 10%	15	66	24	**13**	**57**	**21**
Up 11%	23	93	35	**19**	**72**	**27**
Up 12%	33	116	48	**24**	**88**	**35**
Up 13%	46	161	64	**29**	**103**	**41**
Up 14%	49	220	84	**39**	**115**	**48**
Up 15%	70	209	89	**47**	**120**	**54**
Up 16%	90	233	104	**61**	**134**	**63**
Up 17%	104	249	113	**69**	**142**	**69**
Up 18%	122	296	133	**80**	**150**	**76**
Up 19%	134	312	146	**93**	**161**	**86**
Up 20%	154	350	164	**100**	**169**	**93**
Real Up 6%	**0**	11	3	**0**	**10**	**2**
Real Up 7%	**0**	16	4	**0**	**14**	**3**
Real Up 8%	**0**	20	**6**	**0**	**19**	**6**
Real Up 9%	**2**	38	13	**2**	**25**	**9**
Real Up 10%	9	66	21	**7**	**35**	**13**
Real Up 11%	20	98	36	**14**	**53**	**22**
Real Up 12%	30	133	52	**19**	**73**	**29**
Real Up 13%	40	139	59	**22**	**87**	**34**
Real Up 14%	59	153	70	**28**	**102**	**41**
Real Up 15%	60	196	84	**33**	**105**	**45**
Real Up 16%	93	225	102	**43**	**118**	**53**
Real Up 17%	110	237	110	**51**	**126**	**58**
Real Up 18%	126	263	126	**59**	**138**	**65**
Real Up 19%	136	280	134	**63**	**141**	**68**
Real Up 20%	151	310	152	**70**	**150**	**74**

Table 5: Comparison of the performance of BestInsert and ExactReplan for *building A*. Values marked bold are the best values obtained by any algorithm.

this cannot be avoided, ρ is allocated to the minimum cost elevator anyway. We remark that for the high load up peak situations considered here all algorithms for 2-button systems are essentially the same. The reason is that each elevator will be fully loaded when it leaves the main floor and it has to drop all the passengers before returning. An algorithm for a conventional system has no means to group passengers by destination floors to avoid stopping at many upper floors. This is a major disadvantage of conventional systems [11].

The waiting times for the three systems are reported in Tables 7 and 8. Both destination call systems clearly outperform the conventional system, with the IA system being slightly better for low loads and the DA system performing considerably better for high loads.

The resulting handling capacities are shown in Table 9. In both buildings, a destination call system controlled by ExactReplan increases the handling capacity over that of a conventional system, if only by 1–2%. However, in

Szenario	BestInsert DA system			ExactReplan DA system		
	$\alpha_{0.5}$	$\alpha_{0.9}$	$\emptyset$	$\alpha_{0.5}$	$\alpha_{0.9}$	$\emptyset$
Up 6%	**0**	**8**	**2**	**0**	**8**	**2**
Up 7%	**0**	**10**	**2**	**0**	**10**	**2**
Up 8%	**0**	**13**	**3**	**0**	**13**	**3**
Up 9%	**0**	16	**4**	**0**	**14**	**4**
Up 10%	**0**	20	6	**0**	**19**	**5**
Up 11%	**0**	25	7	**0**	**22**	**6**
Up 12%	**1**	29	9	**1**	**27**	**8**
Up 13%	9	48	17	**6**	**41**	**14**
Up 14%	13	58	21	**9**	**45**	**16**
Up 15%	23	78	31	**14**	**59**	**22**
Up 16%	33	97	40	**18**	**71**	**28**
Up 17%	52	114	53	**26**	**95**	**38**
Up 18%	63	133	64	**30**	**106**	**43**
Up 19%	70	151	71	**32**	**118**	**47**
Up 20%	87	172	85	**42**	**131**	**54**
Real Up 6%	**0**	16	**4**	**0**	**14**	**4**
Real Up 7%	**0**	**15**	**4**	**0**	**15**	**4**
Real Up 8%	**0**	18	**5**	**0**	**17**	**5**
Real Up 9%	**0**	19	6	**0**	**18**	**5**
Real Up 10%	**0**	22	7	**0**	**19**	**6**
Real Up 11%	6	31	11	**4**	**26**	**9**
Real Up 12%	9	38	14	**6**	**27**	**11**
Real Up 13%	12	47	19	**9**	**35**	**14**
Real Up 14%	19	61	25	**12**	**47**	**19**
Real Up 15%	28	84	36	**15**	**60**	**25**
Real Up 16%	40	100	45	**21**	**79**	**33**
Real Up 17%	49	116	53	**24**	**91**	**38**
Real Up 18%	54	129	61	**31**	**103**	**44**
Real Up 19%	67	150	73	**39**	**119**	**52**
Real Up 20%	78	161	79	**44**	**122**	**54**

Table 6: Comparison of the performance of BestInsert and ExactReplan for *building B*. Values marked bold are the best values obtained by any algorithm.

the conventional system elevators are very frequently overloaded, meaning that passengers are left which have to reregister their calls. This can be avoided in destination call systems; thus the effective improvement in service quality is actually more significant. Table 9 also shows that a DA system does not enable a higher handling capacity than an IA system, although a DA system delivers much better waiting times for high loads. The reason is that the service gets unacceptable roughly at the same traffic intensity as for the IA system.

5 Conclusion

In this paper we presented an exact algorithm for the online control of passenger elevator groups. The algorithm is based on a model unifying different elevator control systems and is thus suitable for comparing the performance possible with

Szenario	CGC			ExactReplan IA system			ExactReplan DA system		
	$\alpha_{0.5}$	$\alpha_{0.9}$	$\emptyset$	$\alpha_{0.5}$	$\alpha_{0.9}$	$\emptyset$	$\alpha_{0.5}$	$\alpha_{0.9}$	$\emptyset$
Up 6%	**0**	3	**1**	**0**	**0**	**1**	**0**	1	**1**
Up 7%	**0**	11	3	**0**	**0**	**0**	**0**	10	2
Up 8%	3	22	7	**0**	**6**	**2**	**0**	20	5
Up 9%	20	43	21	**0**	**15**	**4**	3	32	11
Up 10%	39	80	40	**3**	**33**	**11**	13	57	21
Up 11%	53	111	56	**15**	**54**	**21**	19	72	27
Up 12%	75	146	76	30	**87**	**33**	**24**	88	35
Up 13%	97	181	97	43	**102**	45	**29**	103	**41**
Up 14%	116	215	115	58	124	59	**39**	**115**	**48**
Up 15%	129	251	133	61	147	73	**47**	**120**	**54**
Up 16%	155	288	156	72	214	91	**61**	**134**	**63**
Up 17%	174	323	175	79	243	106	**69**	**142**	**69**
Up 18%	192	356	193	90	280	124	**80**	**150**	**76**
Up 19%	214	392	212	126	315	143	**93**	**161**	**86**
Up 20%	233	427	232	135	341	156	**100**	**169**	**93**
Real Up 6%	**0**	15	4	**0**	11	3	**0**	**10**	**2**
Real Up 7%	**0**	16	5	**0**	**13**	**3**	**0**	14	**3**
Real Up 8%	4	26	9	**0**	**18**	**5**	**0**	19	6
Real Up 9%	14	47	18	**0**	**23**	**7**	2	25	9
Real Up 10%	18	71	27	8	38	15	**7**	**35**	**13**
Real Up 11%	44	108	48	22	55	25	**14**	**53**	**22**
Real Up 12%	67	148	71	32	84	36	**19**	**73**	**29**
Real Up 13%	85	178	87	40	102	45	**22**	**87**	**34**
Real Up 14%	95	208	102	53	125	59	**28**	**102**	**41**
Real Up 15%	123	250	127	63	145	71	**33**	**105**	**45**
Real Up 16%	138	271	138	75	169	83	**43**	**118**	**53**
Real Up 17%	144	298	151	85	224	100	**51**	**126**	**58**
Real Up 18%	171	337	173	100	264	118	**59**	**138**	**65**
Real Up 19%	191	362	185	115	279	128	**63**	**141**	**68**
Real Up 20%	212	400	205	123	319	144	**70**	**150**	**74**

Table 7: System performance for *building A*: Waiting times obtained by a conventional system and an IA system and a DA system controlled by ExactReplan. Values marked bold are the best values obtained by any system/algorithm.

Szenario	CGC			ExactReplan IA system			ExactReplan DA system		
	$\alpha_{0.5}$	$\alpha_{0.9}$	$\emptyset$	$\alpha_{0.5}$	$\alpha_{0.9}$	$\emptyset$	$\alpha_{0.5}$	$\alpha_{0.9}$	$\emptyset$
Up 6%	**0**	**7**	**2**	**0**	**7**	**2**	**0**	8	**2**
Up 7%	**0**	**9**	**2**	**0**	10	**2**	**0**	10	**2**
Up 8%	**0**	**12**	**3**	**0**	14	4	**0**	13	**3**
Up 9%	**0**	**14**	**4**	**0**	15	**4**	**0**	**14**	4
Up 10%	**0**	21	6	**0**	**19**	**5**	**0**	**19**	**5**
Up 11%	2	23	8	**0**	**22**	**6**	**0**	**22**	**6**
Up 12%	6	31	11	2	28	9	**1**	**27**	**8**
Up 13%	9	41	15	10	**40**	15	**6**	41	**14**
Up 14%	19	57	24	15	52	20	**9**	**45**	**16**
Up 15%	30	76	34	24	71	30	**14**	**59**	**22**
Up 16%	42	101	44	35	88	38	**18**	**71**	**28**
Up 17%	55	116	56	47	103	52	**26**	**95**	**38**
Up 18%	62	139	67	53	146	66	**30**	**106**	**43**
Up 19%	70	163	77	58	179	75	**32**	**118**	**47**
Up 20%	83	182	90	73	213	96	**42**	**131**	**54**
Real Up 6%	**0**	15	**4**	**0**	16	**4**	**0**	**14**	4
Real Up 7%	**0**	16	5	**0**	**15**	**4**	**0**	**15**	4
Real Up 8%	**0**	18	6	**0**	**17**	**5**	**0**	**17**	**5**
Real Up 9%	1	20	6	**0**	**17**	**5**	**0**	18	**5**
Real Up 10%	3	25	8	**0**	23	8	**0**	**19**	**6**
Real Up 11%	6	30	11	5	28	10	**4**	**26**	**9**
Real Up 12%	13	41	16	8	32	13	**6**	**27**	**11**
Real Up 13%	18	60	24	12	39	16	**9**	**35**	**14**
Real Up 14%	23	60	27	15	50	21	**12**	**47**	**19**
Real Up 15%	36	92	42	26	68	30	**15**	**60**	**25**
Real Up 16%	55	120	58	34	91	42	**21**	**79**	**33**
Real Up 17%	60	134	63	47	104	51	**24**	**91**	**38**
Real Up 18%	61	150	70	49	119	58	**31**	**103**	**44**
Real Up 19%	77	178	87	61	167	76	**39**	**119**	**52**
Real Up 20%	93	194	96	70	185	84	**44**	**122**	**54**

Table 8: System performance for *building B*: Waiting times obtained by a conventional system and an IA system and a DA system controlled by ExactReplan. Values marked bold are the best values obtained by any system/algorithm.

		CGC	ExactReplan	
			IA system	DA system
building A	Up Peak	9%	11%	10%
	Real Up Peak	9%	11%	11%
building B	Up Peak	13%	14%	14%
	Real Up Peak	12%	14%	14%

Table 9: Resulting handling capacities for the considered elevator systems and elevator control algorithms according to Tables 7 and 8.

these systems. The algorithm gives very good schedules after short running time and can therefore be implemented in a real-time environment. In particular, employing our exact algorithm ExactReplan significantly reduces passenger waiting times compared to a state-of-the-art heuristic.

Our work shows that it is worthwhile to investigate exact algorithms for the online control of complex logistics systems, since they can be implemented to deliver high-quality solutions fast. Although the advantage offered by the more flexible DA system and a powerful exact algorithm exploiting this flexibility did not pay off in terms of a higher handling capacity for passenger elevators, similar techniques could be used to improve related systems, e. g., order-picking systems. Moreover, we remark that there are techniques extending this basic version of ExactReplan to increase the handling capacity further. However, these techniques are quite sophisticated and go beyond the scope of this paper and will thus be presented in a follow-up paper.

Acknowledgements We like to thank the SCIP developer team and in particular Gerald Gamrath for advice and explanations to make our Branch & Price algorithm work. Moreover, we thank two anonymous reviewers who made suggestions to improve the paper.

References

[1] Achterberg, T.: SCIP: solving constraint integer programs. Mathematical Programming Computation **1**(1), 1–41 (2009)

[2] Achterberg, T., Berthold, T., Gamrath, G., Heinz, S., Pfetsch, M., Vigerske, S., Wolter, K.: SCIP – solving constraint integer programs. Available at `http://scip.zib.de`

[3] Barney, G.C.: Elevator Traffic Handbook: Theory and Practice. Taylor and Francis (2002)

[4] Desaulniers, G., Desrosiers, J., Solomon, M.M. (eds.): Column generation. Springer (2005)

[5] Friese, P., Rambau, J.: Online-optimization of a multi-elevator transport system with reoptimization algorithms based on set-partitioning models. Discrete Appl. Math. **154**(13), 1908–1931 (2006). Also available as ZIB Report 05-03

[6] Gloss, G.D.: The computer control of passenger traffic in large lift systems. Ph.D. thesis, Victoria University of Manchester (1970)

[7] Hiller, B.: Online optimization: Probabilistic analysis and algorithm engineering. Ph.D. thesis, TU Berlin (2009)

[8] Hiller, B., Klug, T., Tuchscherer, A.: Improving the performance of elevator systems using exact reoptimization algorithms. In: 9th MAPSP (2009)

[9] Hiller, B., Krumke, S.O., Rambau, J.: Reoptimization gaps versus model errors in online-dispatching of service units for ADAC. Discrete Appl. Math. **154**(13), 1897–1907 (2006). Traces of the Latin American Conference

on Combinatorics, Graphs and Applications – A selection of papers from LACGA 2004, Santiago, Chile

[10] Hiller, B., Tuchscherer, A.: Real-time destination-call elevator group control on embedded microcontrollers. In: Operations Research Proceedings 2007, pp. 357–362. Springer (2008)

[11] Hiller, B., Vredeveld, T.: Stochastic dominance analysis of online bin coloring algorithms. ZIB Report 12-42, Zuse Institute Berlin (2012). `http://opus4.kobv.de/opus4-zib/frontdoor/index/index/docId/1650`

[12] Koehler, J., Ottiger, D.: An AI-based approach to destination control in elevators. AI Magazine **23**(3), 59–78 (2002)

[13] Krumke, S.O., Rambau, J., Torres, L.M.: Realtime-dispatching of guided and unguided automobile service units with soft time windows. In: Proceedings of the 10th ESA, *LNCS*, vol. 2461, pp. 637–648. Springer (2002)

[14] Schröder, J.: Advanced dispatching: Destination hall calls + instant car-to-call assignments: M10. Elevator World pp. 40–46 (1990)

[15] Seckinger, B., Koehler, J.: Online-Synthese von Aufzugssteuerungen als Planungsproblem. In: 13th Workshop on Planning and Configuration, pp. 127–134 (1999)

[16] Smith, R., Peters, R.: ETD algorithm with destination dispatch and booster options. Elevator World pp. 136–145 (2002)

[17] Tanaka, S., Uraguchi, Y., Araki, M.: Dynamic optimization of the operation of single-car elevator systems with destination hall call registration. European J. Oper. Res. **167**(2), 550–587 (2005)

[18] Tyni, T., Ylinen, J.: Evolutionary bi-objective optimisation in the elevator car routing problem. European J. Oper. Res. **169**(3), 960–977 (2006)