A Column Generation Approach for Solving the Patient Admission Scheduling Problem

Range, Troels Martin ; Lusby, Richard Martin ; Larsen, Jesper

Publication date:
2013

Document Version
Publisher's PDF, also known as Version of record

A Column Generation Approach for Solving the Patient Admission Scheduling Problem

by

Troels Martin Range,

Richard Martin Lusby

and

Jesper Larsen

Discussion Papers on Business and Economics
No. 1/2013

FURTHER INFORMATION
Department of Business and Economics
Faculty of Social Sciences
University of Southern Denmark
Campusvej 55
DK-5230 Odense M
Denmark

Tel.: +45 6550 3271
Fax: +45 6550 3237
E-mail: lho@sam.sdu.dk
http://www.sdu.dk/ivoe

ISBN 978-87-91657-79-5

A Column Generation Approach for Solving the Patient Admission Scheduling Problem

Troels Martin Range

Department of Business & Economics, COHERE, University of Southern Denmark,
Campusvej 55, 5230 Odense M, Denmark,
tra@sam.sdu.dk

Richard Martin Lusby

Department of Engineering Management, Technical University of Denmark
Produktionstorvet, building 426, 2800 Kgs. Lyngby
rmlu@dtu.dk

Jesper Larsen

Department of Engineering Management, Technical University of Denmark
Produktionstorvet, building 426, 2800 Kgs. Lyngby
jesla@dtu.dk

January 8, 2013

Abstract

This paper addresses the Patient Admission Scheduling (PAS) problem. The PAS problem deals with assigning elective patients to beds, satisfying a number of soft and hard constraints. The problem can be seen as part of the functions of hospital management at an operational level. There exists a small number of different variants on this problem. We propose an optimization-based heuristic building on branch-and-bound, column generation, and dynamic constraint aggregation for one of the variants. We achieve tighter bounds than previously reported in the literature, and in addition we are able to produce new best solutions for five out of six instances from a publicly available repository.

Keywords: Patient admission scheduling, column generation, dynamic constraint aggregation, dual disaggregation, branch-and-bound.
JEL Code: C61 **MSC Code:** 90B35, 90C57

1 Introduction

In this paper we consider a new approach for solving the Patient Admission Scheduling (PAS) problem. In many countries, the allocation of patients to wards (more specifically, to

beds) is carried out by a central planning unit in the hospital management. The algorithm presented in this paper can assist hospital management at the operational level in assigning elective patients to beds, attempting to satisfy as many as possible of the individual patient preferences, while making sure critical medical equipment is available. Hospitals usually have slack in order to deal with emergency patients, which are, naturally, difficult to plan for. Emergency patients are not considered in this study.

The area of patient admission scheduling in hospitals is important for a high-quality health care system. To provide health care services is perhaps one of the most complex industries worldwide. Planning and managing the operations of a hospital in an efficient manner requires a sound knowledege of the hospital system, and understanding the patient flow is critical for this. Most literature within patient admission scheduling focuses on the strategic (and tactical) level and describes tools and cases for assisting hospital management with tactical and strategic planning problems. Some examples are Jittamai and Kangwansura [2011], Harper [2002], Kusters and Groot [1996]. The aim here is to have the right number of beds available in order to increase the efficiency of the hospital.

The problem that we investigate in this paper is the operational version of the PAS problem. This problem has not received much attention until recently. It is indirectly part of scheduling operating rooms, which has achieved some attention. The survey in Guerriero and Guido [2011] lists some 129 publications that focus on optimization of the operating theaters of a hospital.

The operational version of PAS has been described with slightly different twists in the few existing articles available. In Hutzschenreuter et al. [2008] an agent-based model for the problem is presented. A model of the patient flow is generated in order to admit an optimal mix of patients from different departments with special attention to the common usage of (central) hospital resources. The work is based on a detailed description of the patient flow and its probabilities. This system can be used to develop policies for achieving a good patient mix and to facilitate efficient hospital operations.

In contrast to this, Chen et al. [2010] develop a generic algorithm for an admission scheduling problem for only one department. The algorithm uses historical data and optimizes a long-term admission strategy instead of suggesting a specific schedule.

The remaining work in this area, Demeester et al. [2010], Bilgin et al. [2012], Ceschia and Schaerf [2011], all consider the same version of PAS. This definition of the PAS problem was originally presented in Demeester et al. [2010]. In addition, the authors have published a website containing instances[1], currently best known solutions, and a solution validator.

The problem has a fixed number of elective patients. In practice, elective patients often wait for an admission date; however, in this particular context, each patient has already been assigned both an admission and a discharge date. In addition, each patient states their medical requirements. The aim is then to determine which beds are assigned to which patients in order to maximize the patient comfort and efficiency of the medical operations. This planning has to be done in compliance with a few hard constraints; for example, admission and discharge dates must be respected, mandatory medical equipment needs to be in the room assigned, and male and female patients cannot share the same room. In this paper we present a new mathematical formulation of the problem and devise an efficient column generation-based heuristic for solving it. Our methodology utilizes a dynamic constraint aggregation procedure to overcome some of the problems associated with solving large integer programming formulations. The methodology is tested on the first six of the benchmark in-

[1] http://allserv.kahosl.be/∼peter/pas.

stances provided by Demeester et al. [2010], where we report tighter LP bounds, signficantly faster running times than previous mathematical programming approaches to achieve these, and five new best known integer solutions.

The structure of this paper is as follows. In Section 2, we introduce the PAS problem in more detail and present a binary integer programming formulation to model it. Section 3 describes the proposed solution methodology as well as a detailed discussion on the dynamic constraint aggregation procedure. Here we also introduce a new dual disaggregation strategy. Computational results are described in Section 4, and conclusions from this study are drawn in Section 5.

2 Problem Definition

In this section we consider the PAS problem in more detail. In particular, we provide a detailed overview of the constraints of the problem and introduce the required terminology and notation that is used throughout the paper. The description of the problem is consistent with that of Demeester et al. [2010].

The PAS problem requires that a set of patients requiring medical attention, $\mathcal{P}$, are assigned to a set of hospital beds over a prespecified daily time horizon. We denote the set of consecutive days that comprise this planning horizon as $\mathcal{T}$. Each patient, $p \in \mathcal{P}$, is assumed to have a known admission date, $a_p \in \mathcal{T}$, and a known discharge date, $d_p \in \mathcal{T}$, which together define the duration of the patient's stay in the hospital. We let $\mathcal{W} = \{(p,t) \in \mathcal{P} \times \mathcal{T} | a_p \leq t < d_p\}$ be the set of all *patient-time combinations*; in other words, the set of all patient days which must be assigned at the hospital.

For treatment each patient requires one or more so-called *specialisms* (e.g. cardiology, oncology, gerontology, etc.). Typically, patients need just one specialism. There are, however, some *multi-spec* patients who require more than one. In this case, the time needed by each specialism is known. Each bed is located in a particular *room*, and each room belongs to a particular *department* at the hospital. Every department has the ability to cater for the treatment of a variety of specialisms, but with varying degrees of expertise. Some departments also have an *age policy* (e.g. pediatrics, gerontology), which means that the department can only admit patients of a certain age. As is the case for departments, rooms also have a ranked list of specialisms for which they are suitable. Furthermore, every room has a certain number of identical beds, termed the *room capacity* (typically this is one, two, or four beds) and is equipped with a set of *properties* (e.g. oxygen and telemetry) that can be used for treatment. The presence of certain room properties for a patient may either be *required* or simply *preferred*. Each room also has a *gender policy*. This stipulates that all patients in the room must be male, female, mixed (but the same gender on any given day), or unrestricted. The rooms at the hospital can be classified as a particular *room type*, where all rooms of a particular type are identical. We denote the set of all room types at the hospital as $\mathcal{R}$, and for each $r \in \mathcal{R}$ we let N_r be the number of available rooms of this type. Since all beds in a given room are identical and all the rooms of a particular type are also identical, one can equivalently view the PAS problem as finding an assignment of patients to room types over the planning horizon.

In the process of assigning patients to room types, several constraints must be respected. These can be classified as *hard* or *soft* constraints. The former must be respected, while the latter can be violated; however, a penalty is incurred for each violation. For the version of the PAS problem that we consider, there are three types of hard constraints. Firstly, there

is the rather obvious requirement that no two patients can simultaneously occupy the same bed. Viewed from a room type assignment perspective, this is equivalent to stating that the capacity of the room type must not be violated on any day. Secondly, on each day of the planning horizon each room type should contain patients that correspond to its gender policy. Finally, each patient should be assigned a room type consistent with their age.

All other constraints are considered soft and will be described in turn. We begin by discussing those penalties that are incurred when assigning a given patient, $p \in \mathcal{P}$, to a particular room type, $r \in \mathcal{R}$, since all such individual penalties can be accumulated to obtain one patient room type assignment penalty, d_{pr}. Firstly, each patient has a preference for the capacity of the room in which they will stay. A patient who is assigned a room type with a capacity larger than their preference incurs a penalty. Secondly, a patient should be placed in a department whose most competent specialism matches the patient's required specialism. Lower levels of expertise are penalized. Similarly, a patient should receive a room type that best matches the patient's specialism. Imperfect matches are again penalized. Finally, in terms of room properties, patients should receive a room type that is equipped with their required and preferred properties. Missing properties are penalized; however, missing required properties are more heavily penalized than preferred ones. The only penalty not related to room type assignment concerns the transfer of patients between room types on successive days. Since it is undesirable to transfer patients during their stay, every time a patient is transferred a transfer-out penalty, t^o, and a transfer-in penalty, t^i, are incurred.

Given the problem description outlined above, one can formally state the PAS problem as follows: find the minimum penalty allocation of patients to room types so that each patient is allocated to a room type during their stay while respecting the various room type capacities, gender policies, and age policies.

Figure 1 gives an illustrative example of how patients could be assigned to a room type containing four beds over a 14-day time horizon. Male patients are indicated by a solid gray rectangle, while female patients are represented by a dark gray, hatched rectangle. The horizontal lines connecting successive rectangles indicate that the respective rectangles represent the same patient. A solid white rectangle simply indicates that the relevant bed is empty. Note that this particular assignment of patients over the planning horizon corresponds to a *room schedule* for this room type. The set of all feasible room schedules for a given room type, $r \in \mathcal{R}$, is denoted as $\mathcal{S}_r$.

The PAS problem can be formulated as the following 0-1 integer program. The binary decision variable x_{rs} states whether or not the solution contains room schedule $s \in \mathcal{S}_r$ for room type $r \in \mathcal{R}$. Allocating room schedule $s \in \mathcal{S}_r$ to room type $r \in \mathcal{R}$ is assumed to incur c_{rs} in penalties (i.e. the sum of all penalties incurred for the room type over the time horizon). The binary indicator variable a_{sw} states whether or not a given room schedule, $s \in \mathcal{S}_r$, contains patient time combination $w \in \mathcal{W}$.

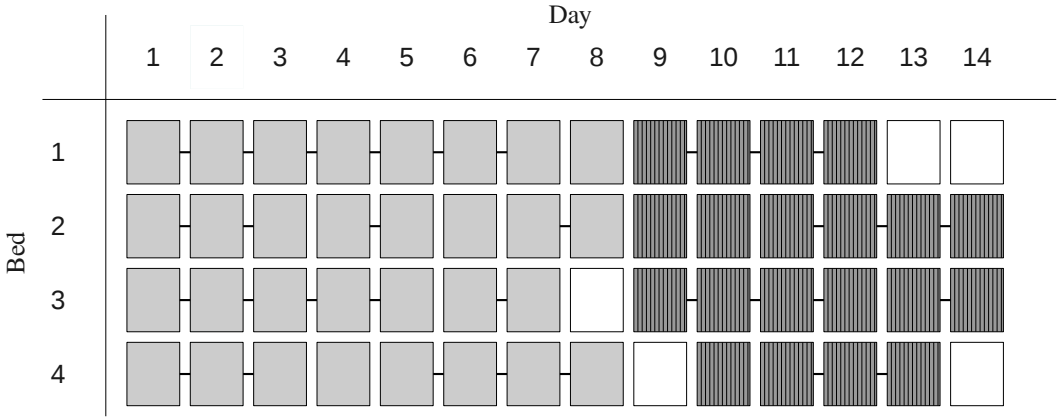

Figure 1: Example of a room schedule for 14 days

$$\min \quad \sum_{r \in \mathcal{R}} \sum_{s \in \mathcal{S}_r} c_{rs} x_{rs}, \tag{1}$$

$$\text{s.t.} \quad \sum_{r \in \mathcal{R}} \sum_{s \in \mathcal{S}_r} a_{sw} x_{rs} = 1 \qquad \forall w \in \mathcal{W}, \tag{2}$$

$$\sum_{s \in \mathcal{S}_r} x_{rs} \leq N_r \qquad \forall r \in \mathcal{R}, \tag{3}$$

$$x_{rs} \in \{0, 1\} \qquad \forall r \in \mathcal{R}, \forall s \in \mathcal{S}_r. \tag{4}$$

The objective function, given by (1), minimizes the sum of the penalties incurred in allocating the room schedules. Constraint set (2) ensures that each patient is assigned a room for each night of their respective stays, while constraints (3) prevent staff from assigning more room schedules for a particular room type than there are rooms of that type. The binary restrictions on the decision variables are given by (4). To complete the description we let π_w be the dual variable value on constraint (2) associated with patient-time combination $w \in \mathcal{W}$ and let σ_r be the dual variable value on constraint (3) associated with room type $r \in \mathcal{R}$.

It may be observed that this formulation is essentially from a room type perspective. That is, only legal room schedules for each room type are included in the model. As a consequence, the constraints on the gender policy of the room type and its bed count are embedded in the column definition of the model. Therefore, we expect the LP relaxation of this formulation to be tighter than one that simply considers the assignment of patients to beds. For the latter case, additional constraints are required to enforce both the gender requirement and the bed counts of the respective room types.

The above formulation is characterized by an exponential number of variables and, comparatively speaking, relatively few constraints. This makes it an ideal candidate for a column generation based approach. However, while the number of constraints is far smaller than the number of possible variables, it is still large enough ($|\mathcal{W}|$ is at least 2244 for the instances we consider) to impede the performance of a linear program solver. Hence, we develop a column generation approach which utilizes dynamic constraint aggregation to help reduce

the row dimension of the formulation. The details of this algorithm will be described in the subsequent section.

3 Solution Approach

To solve the PAS problem we propose a column generation-based approach that incorporates a dynamic constraint aggregation procedure to improve the performance of the algorithm. The dynamic constraint aggregation is of crucial importance in that it reduces the row dimension of the set partitioning problem. Column generation is a widely used decomposition technique for solving large-scale optimization problems in which it is impossible to consider all variables explicitly in the formulation. In column generation the problem is decomposed into a *master problem* and one (or more) *pricing problem(s)*, respectively. In what follows we describe the structure of both the master and subproblems as well as provide a detailed explanation of how we apply the dynamic constraint aggregation methodology.

3.1 Master Problem

The PAS master problem is identical in structure to that of model (1)-(4); however, it only contains a subset of the possible variables. The role of the pricing problem is to identify favorable variables dynamically using the dual variable information from an optimal solution of the *relaxed* master problem. To obtain the relaxed master problem, we replace constraints (4) with

$$x_{rs} \geq 0. \tag{5}$$

The premise in column generation is that many of the variables will be non-basic in an optimal solution to the original problem, so one should only generate those variables that have the potential to improve the objective function value. Thus, the objective function for the pricing problem is the reduced cost calculation of the non-basic variables. For the PAS problem we identify one pricing problem for each room type, where each has an objective function of the form

$$\sum_{(p,t)\in\mathcal{W}} (d_{pr} - \pi_{pt}) z_{tp} + t^i \gamma^i + t^o \gamma^o - \sigma_r. \tag{6}$$

Here, z_{tp} is a binary variable indicating whether or not patient time combination $(p,t) \in \mathcal{W}$ is assigned to room type $r \in \mathcal{R}$, and γ^i and γ^o give the total number of transfers into and out of room type $r \in \mathcal{R}$ over the planning horizon.

Column generation is hence an iterative procedure between the master and pricing problems. The master problem solves to optimality a relaxed, restricted version of the original problem, which contains a subset of promising variables, while the pricing problems identify favorable variables to append to the master problem. If no such variable is found, the process terminates, and one has found an optimal solution to the LP relaxation of the original problem. If integrality is desired, column generation can be embedded in a branch-and-price framework. The reader is referred to Desrosiers et al. [2005] for an introduction to column generation.

3.2 Pricing Problem

In the pricing problem we have to identify a room schedule for each room type satisfying the room constraints such that (6) is minimized. That is, a room schedule that places at most Q_r patients in the room in any time period and satisfies the gender constraints.

We let $\mathcal{P}_r \subseteq \mathcal{P}$ be the subset of patients which are compatible with room specification, thereby leaving out patients who do not satisfy the gender policy or the age policy for the room. We let $\mathcal{V}_r = \{(p,t) \in \mathcal{W} | p \in \mathcal{P}_r\}$ be the considered patient-time combinations for the room type r.

The gender policy of the room can be considered as a categorization of the patients. If the room has either the male policy, the female policy or the unrestricted policy, then the room only has a single category of patients. On the other hand, if the room has the mixed policy, then both genders are considered, but only one of the genders can be present in the room at any time. Thus, let C be the index set of the categories and let $m_{cp} \in \{0,1\}$ be a parameter taking the value 1 if and only if patient $p \in \mathcal{P}_r$ belongs to category $c \in C$.

A variable γ^i_{pt} measures whether or not patient p is transferred into the room from period $t-1$ to period t, and γ^o_{pt} indicates whether or not patient p is transferred out of the room from period $t-1$ to period t. For each day t the variable v_{ct} states whether or not a patient from category c is present in the room. A mathematical model for the pricing problem is then

$$\min \quad \sum_{(p,t)\in\mathcal{V}_r} (d_{pr} - \pi_{pt}) z_{pt} + \sum_{(p,t)\in\mathcal{V}_r} (t^i \gamma^i_{pt} + t^o \gamma^o_{pt}) \tag{7}$$

$$\text{s.t.} \quad \sum_{p\in\mathcal{P}_r} z_{pt} \leq Q_r, \qquad t \in \mathcal{T} \tag{8}$$

$$z_{pt} - z_{p,t-1} \leq \gamma^i_{pt} \qquad p \in \mathcal{W}_r, a_p < t < b_p \tag{9}$$

$$z_{p,t-1} - z_{pt} \leq \gamma^o_{pt} \qquad p \in \mathcal{W}_r, a_p < t < b_p \tag{10}$$

$$m_{cp} z_{pt} \leq v_{ct} \qquad c \in C, t \in \mathcal{T} \tag{11}$$

$$\sum_{c\in C} v_{ct} \leq 1 \qquad t \in \mathcal{T} \tag{12}$$

$$z_{pt} \in \{0,1\} \qquad (p,t) \in \mathcal{V}_r \tag{13}$$

$$\gamma^i_{pt}, \gamma^0_{pt} \geq 0 \qquad (p,t) \in \mathcal{V}_r \tag{14}$$

$$v_{ct} \geq 0 \qquad c \in C, t \in \mathcal{T}. \tag{15}$$

The objective (7) corresponds to the variable part of the reduced cost coefficient of a column given in (6). The number of patients transferred into the room is $\gamma^i = \sum_{(p,t)\in\mathcal{V}_r} \gamma^i_{pt}$ and the number of patients transferred out of the room is $\gamma^o = \sum_{(p,t)\in\mathcal{V}_r} \gamma^o_{pt}$. Constraints (8) ensure that the room capacity is not violated. For each patient in each period within the patient's admission, constraints (9) and (10) measure, respectively, whether or not the patient is transferred into the room and whether or not the patient is transferred out of the room. Next, constraints (11) and (12) enforce the gender policy where only one category, as defined above, is allowed into the room in each period. If we only have a single category, then $|C| = 1$ and constraints (11) and (12) will be redundant, in which case they can be removed from the problem. Finally, the types of variables are stated in (13)-(15). Note that if the z_{pt}-variables are binary, then the other variables will take binary values in an optimal solution, and thus we let the other variables be standard non-negative continuous variables.

Instead of solving model (7)-(15) directly, we use a variety of heuristics and dynamic programming algorithms to solve the pricing problems. We use heuristics as, most of the time, they yield sufficiently good columns with negative reduced cost using only short computation time. When the heuristics cannot identify any negative reduced cost columns, we turn to an exact dynamic programming approach, which gives a number of negative reduced cost columns if any such columns exist.

We primarily use three different pricing heuristics. The two first are based on greedily selecting patients for their full stay by increasing contribution to the reduced-cost coefficient. In this way, we avoid the penalties of transferring patients in and out of the room. The first heuristic greedily selects the patient with most negative contribution to the reduced cost which is compatible with the already allocated patients. In this heuristic, patients are not selected according to the time interval during which they occupy the bed and, as a consequence, the corresponding schedule can display unused beds on many days. To remedy this, the second heuristic increments time period, t, such that we can only select patients, p, with $a_p = t$. Then patients starting in period t are selected greedily as long as they are compatible with already allocated patients. If no more patients can be found, then the period t is incremented and the process is repeated.

The third heuristic is based on the following observation. If the room has only one bed, i.e. capacity $Q_r = 1$, then the pricing problem reduces to a shortest-path problem in a directed acyclic graph. Hence, solving this shortest path problem and multiplying the resulting value with the room capacity yield a lower bound on the problem (7)-(15). Subtracting σ_r from this lower bound gives a lower bound on the reduced-cost coefficient of any schedule for room type r. On the other hand, if we resolve the shortest-path problem, where we have removed the patients not compatible with the patients found on previous shortest paths, then we will get a feasible room schedule, where each path corresponds to the occupation of a single bed in the room throughout the planning period. Thus, solving the increasingly restricted shortest path problem Q_r times results in a feasible room schedule. In contrast to the two first greedy heuristics, the shortest path based heuristic allows transfers.

The two first heuristics are very fast and are both run for each room in each pricing. The shortest path-based heuristic is run only for those room types which have a negative lower bound on the reduced costs.

When the heuristics cannot identify any negative reduced cost room schedules, we solve the pricing for a subset of room types by dynamic programming. The sequence in which we try to price out each room type is based on the lower bound on the reduced cost for any schedule for the room type. We select the room types in increasing value of the lower bound of the reduced cost. If the number of negative reduced cost columns obtained is sufficiently large, then the exact pricing is prematurely terminated. If not, then the dynamic programming algorithm is run on all room types having a negative lower bound on the reduced cost.

The dynamic programming approach uses the increasing days $t \in \mathcal{T}$ as stages and $(C_t(S), S)$ as states, where $S \subseteq \mathcal{P}_r$ having $|S| \leq Q_r$ and the patients in S are compatible with each other, i.e. do not violate the gender policy. $C_t(S)$ is then the lowest penalty with which we can obtain S in period t using any feasible sequence of sets $S_1, \ldots, S_{t-1}$. Given two sets, S' and S, we let $\gamma^i_{t-1,t}(S', S)$ be a function calculating the number of patients transferred in from S' to S. Likewise, we let $\gamma^o_{t-1,t}(S', S)$ be the number of patients transferred out of the room from S' to S. Note that these two functions do not add newly arrived patients or discharged patients. If we let $\mathcal{F}_t$ be the set of feasible subsets $S \subseteq \mathcal{P}_r$ satisfying 1) that for each patient $p \in S$ the patient has to be admitted, i.e. $a_p \leq t < b_p$;

2) that the number of patients in the room at time t is no larger than the number of beds available, i.e. $|S| \leq Q_r$; and 3) that the gender policy of the room is satisfied. Then we can calculate $C_t(S)$ by the following recursion:

$$C_t(S) = \begin{cases} \sum_{p \in S}(d_{pr} - \pi_{pt}), & t = 1 \\ \sum_{p \in S}(d_{pr} - \pi_{pt}) + \min_{S' \in \mathcal{F}_{t-1}} \left\{ C_{t-1}(S') + t^i \gamma^i_{t-1,t}(S', S) + t^o \gamma^o_{t-1,t}(S', S) \right\} & t > 1 \end{cases} \tag{16}$$

In each stage the dynamic programming constructs the sets S and identifies the cost by the recursion (16). The size of $\mathcal{F}_t$ is exponential in Q_r and $|\mathcal{P}_r|$, and thus this algorithm is not polynomially bounded in the worst case. We can, however, apply preprocessing as well as dominance criteria between states, which significantly reduces the computation times. The preprocessing is stronger the closer the solution of the pricing problem is to price out, which makes it possible to solve the dynamic programming problem for a single room in less than a second in most cases. The reader is referred to Range et al. [2012] for further details on the preprocessing, the dynamic programming algorithm as well as the shortest path based bound.

3.3 Dynamic Constraint Aggregation

Dynamic constraint aggregation is a technique for efficiently solving large set partitioning problems. Set partitioning problems with a large row dimension are notoriously degenerate, and this often results in excessively long solution times for the resulting linear programs. Dynamic constraint aggregation attempts to reduce the row dimension (and hence the potential degeneracy) of set partitioning problems by partitioning the constraints into aggregated sets of constraints, known as *clusters*. The set partitioning problem is then restated in terms of the clusters; that is, there is one partitioning constraint for each cluster. The reduced size of the aggregated problem means it is also easier to solve.

Naturally, the aggregation impacts the columns that can be included in the model. Only columns that are *compatible* with the aggregation are considered in the optimization. A column is said to be compatible if it covers all elements of a cluster or none of them. Due to the possibility of excluding favorable columns depending on the choice of aggregation, the aggregation is dynamically updated if one detects incompatible columns that have a favorable reduced cost. A theoretical framework for this methodology was first proposed by Villeneuve [1999], while the first implementation of dynamic constraint aggregation can be found in Elhallaoui et al. [2005]. Elhallaoui et al. [2010] make the procedure so-called *multi phase*. In the multi phase approach a strategy for pricing columns in order of increasing incompatibility is incorporated.

According to Elhallaoui et al. [2005], the initial motivation for constraint aggregation arose in crew scheduling applications, where it was observed that crews very rarely change vehicles. A similar property is also true for the PAS problem: here the transferring of patients is extremely undesirable. That is, all patient time combinations for a single patient are likely to be assigned to the same room schedule. Using this observation, one could consider all consecutive patient-time combinations for a certain patient as a single, aggregated patient-time combination. Formally, in patient admission scheduling, an aggregation of the partitioning constraints into a set of clusters $\mathcal{H}$ results in a partition, Q, of $\mathcal{W}$. This is defined as follows:

$$Q = \{\mathcal{W}_h : h \in \mathcal{H}\}, \text{ where } \mathcal{W}_i \cap \mathcal{W}_j = \emptyset \; \forall i, j \in \mathcal{H}, i \neq j, \bigcup_{h \in \mathcal{H}} \mathcal{W}_h = \mathcal{W}$$

.

3.3.1 Aggregated Master Problem

The aggregated master problem can be written as follows. Here all notation is the same as the disaggregated model with the exception of the indicator variables a_{sh}. These simply indicate whether or not a given room schedule, $s \in \mathcal{S}_r$ ($r \in \mathcal{R}$), covers all the patient-time combinations contained in partition $h \in \mathcal{H}$ or none of them.

$$\min \quad \sum_{r \in \mathcal{R}} \sum_{s \in \mathcal{S}_r} c_{rs} x_{rs}, \tag{1}$$

$$\text{s.t.} \quad \sum_{r \in \mathcal{R}} \sum_{s \in \mathcal{S}_r} a_{sh} x_{rs} = 1 \qquad \forall h \in \mathcal{H}, \tag{17}$$

$$\sum_{s \in \mathcal{S}_r} x_{rs} \leq N_r \qquad \forall r \in \mathcal{R}, \tag{3}$$

$$x_{rs} \geq 0 \qquad \forall r \in \mathcal{R}, \forall s \in \mathcal{S}_r. \tag{5}$$

Observe that the objective function (1), constraints (3), and constraints (4) are inherited from the disaggregated model. The only difference arises in constraint set (17). Here there is now just one partitioning constraint for each partition $h \in \mathcal{H}$. To complete the description, we let α_h be the dual variable value on constraint (17) associated with partition $h \in \mathcal{H}$. Since constraint set (3) is unchanged from the disaggregated model, the dual variable on constraint (3) associated with room type $r \in \mathcal{R}$ remains σ_r.

3.3.2 Dual Disaggregation

In order to generate new columns that are compatible with the disaggregated model (or prove that none exists), one must first obtain duals that are consistent with the disaggregated model by disaggregating the dual variable values obtained from the aggregated master problem. Elhallaoui et al. [2005] disaggregate duals by solving shortest-path problems in a specially designed graph. They observe that the dual has to satisfy

$$\sum_{w \in \mathcal{W}_h} \pi_w = \alpha_h, \quad \forall h \in \mathcal{H} \tag{18}$$

in order for the disaggregated dual variable values to be consistent with the aggregated dual variable values. To further disaggregate the aggregated duals, the authors apply a heuristic based on a shortest-path problem where as many of the dual constraints for the incompatible columns are satisfied. The main reason for using the heuristic is that it is reasonably fast.

Here we present a new dual disaggregation strategy that is similar in structure to that of the linear programming model proposed by Elhallaoui et al. [2011] for solving the so-called *complementarity problem*. This problem arises in the author's improved primal simplex algorithm and requires one to obtain a complete, feasible dual solution from a reduced problem. Instead of a complementarity problem we solve a dual feasibility problem directly. When applying dynamic constraint aggregation, we observe that only a small subset of

clusters is actually violated. A violated cluster is one for which there exists an incompatible room schedule that covers some, but not all, of the patient-time combinations contained in the cluster. In what follows we denote the subset of non-violated (resp. violated) clusters as $\mathcal{U} \subseteq \mathcal{H}$ (resp. $\overline{\mathcal{U}} \subseteq \mathcal{H}$). Furthermore, we denote the set of room schedules for a certain room type, $r \in \mathcal{R}$, that are incompatible with the current aggregation as $\overline{\mathcal{S}}_r^Q$. Each of the compatible room schedules is included in the aggregated master problem and all relevant information that can be derived from these is contained in the aggregated dual α_u, where $u \in \mathcal{U}$. As these aggregated duals can be disaggregated in any way such that (18) is feasible, the only columns of interest when disaggregating the aggregated dual variables are those representing incompatible room schedules. A dual constraint for any room schedule can be written as

$$\sum_{h \in \mathcal{H}} \sum_{w \in \mathcal{W}_h} a_{sw}\pi_w + \sigma_r \leq c_{rs} \quad \forall r \in \mathcal{R}, \quad \forall s \in \overline{\mathcal{S}}_r^Q.$$

If we rewrite this in terms of the sets $\mathcal{U}$ and $\overline{\mathcal{U}}$ and move the constant terms to the right-hand side, we obtain

$$\sum_{u \in \overline{\mathcal{U}}} \sum_{w \in \mathcal{W}_u} a_{sw}\pi_w \leq c_{rs} - \sigma_r - \sum_{u \in \mathcal{U}} \sum_{w \in \mathcal{W}_u} a_{sw}\pi_w \quad \forall r \in \mathcal{R}, \quad \forall s \in \overline{\mathcal{S}}_r^Q.$$

One knows by construction that no room schedule violates any cluster $u \in \mathcal{U}$. Thus, one can replace $\sum_{u \in \mathcal{U}} \sum_{w \in \mathcal{W}_u} a_{sw}\pi_w$ with $\sum_{u \in \mathcal{U}} a_{su}\alpha_u$ and obtain

$$\sum_{u \in \overline{\mathcal{U}}} \sum_{w \in \mathcal{W}_u} a_{sw}\pi_w \leq c_{rs} - \sigma_r - \sum_{u \in \mathcal{U}} a_{su}\alpha_u \quad \forall r \in \mathcal{R}, \quad \forall s \in \overline{\mathcal{S}}_r^Q. \tag{19}$$

These constraints must be satisfied if the aggregated dual solution is an optimal dual solution to the disaggregated master problem. To ease the notation, we let $y_{rs} = c_{rs} - \sigma_r - \sum_{u \in \mathcal{U}} a_{su}\alpha_u$ and formulate the following LP to determine if the aggregated dual variables can be successfully disaggregated.

$$\min \quad \sum_{r \in \mathcal{R}} \sum_{s \in \overline{\mathcal{S}}_r^Q} s_{rs}, \tag{20}$$

$$\text{s.t.} \quad \sum_{w \in \mathcal{W}_u} \pi_w = \alpha_u \qquad \forall u \in \overline{\mathcal{U}}, \tag{21}$$

$$\sum_{u \in \overline{\mathcal{U}}} \sum_{w \in \mathcal{W}_u} a_{sw}\pi_w - s_{rs} \leq y_{rs} \qquad \forall r \in \mathcal{R}, s \in \overline{\mathcal{S}}_r^Q, \tag{22}$$

$$s_{rs} \geq 0 \qquad \forall r \in \mathcal{R}, s \in \overline{\mathcal{S}}_r^Q, \tag{23}$$

$$\pi_w \geq 0 \qquad \forall w \in \mathcal{W}. \tag{24}$$

Here, s_{rs} (where $r \in \mathcal{R}, s \in \overline{\mathcal{S}}_r^Q$) are auxiliary variables capturing the magnitude of violation for the relevant dual constraint (19). The objective function, given by (20), minimizes the sum of the violations. Constraintset (21) enforces the requirement that the sum of the disaggregated dual variables is the value of the aggregated dual, while constraints (22) determine the magnitude of violation in constraints (19). Constraints (23) are the non-negativity requirements on the s_{rs} variables. Finally, we also add constraints (24) to ensure that the disaggregated dual variables are also non-negative. This is because we relax the

partitioning constraints (17) to be covering constraints when solving the aggregated master problem and thus restrict the dual variables to be non-negative, effectively stabilizing them. By restricting the sign of the aggregated dual variables α_h ($h \in \mathcal{H}$) to be non-negative, we can automatically set $\pi_w = 0$ for $w \in \mathcal{W}_h$ whenever $\alpha_h = 0$ ($h \in \mathcal{H}$). This stabilizing technique gave a significant speed up when solving the dual disaggregation problem.

If one obtains an objective function value of zero when solving this problem, dual disaggregation is indeed feasible and the solution can be used to price new columns; however, if the converse is true, i.e. if there is at least one room schedule for which the corresponding $s_{rs} > 0$, then the aggregated dual solution can never be optimal for the disaggregated problem and re-aggregation is necessary. When reaggregating, room schedules with $s_{rs} > 0$ are forced into the aggregation to ensure that they are compatible with the aggregation. An overview of the solution approach is depicted in Figure 2.

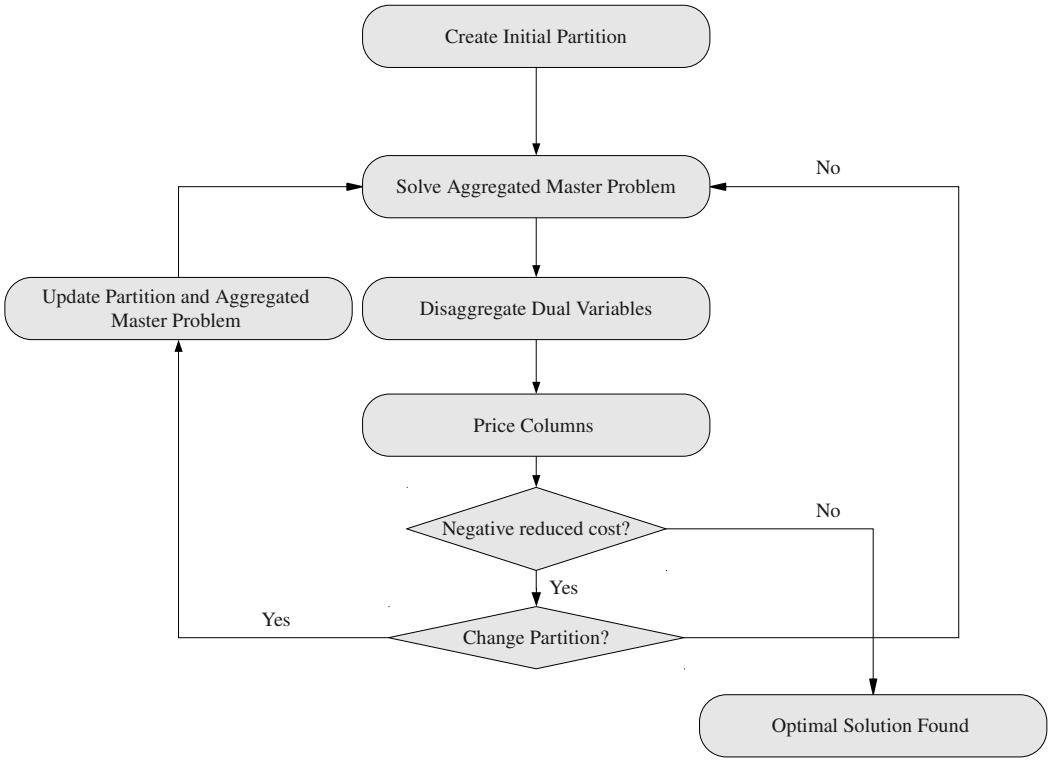

Figure 2: Dynamic Constraint Addition

As a starting point, the algorithm requires one to construct an initial aggregation. To do this, we implement a greedy, daily assignment heuristic with look-ahead penalties to yield an upper bound. The initial aggregation is based on this heuristic solution. At this point the algorithm enters an iterative loop betwen the master problem and room type subproblems. If no subproblem returns a room schedule with negative reduced cost, the current set of room schedules in the relaxed, aggregated master problem is optimal. Otherwise, a decision of whether the aggregation should be revised or not is made. Typically, the decision to reaggregate is based on the number of incompatible columns returned from the pricing problems. If only incompatible columns are returned, reaggregation is necessary, while if

only compatible columns are returned, no change to the aggregation is necessary. In the case where both incompatible and compatible columns are returned a decision is usually based on the ratio of the best reduced cost from each set.

3.4 Forcing Integrality

The previous sections describe a column generation based approach for solving the relaxed master problem. However, to find a solution to the original problem, i.e. model (1)-(4), one must reintroduce the binary restriction on the decision variables, where it is violated. In this section we describe possible strategies for enforcing the integrality of the solution. Finally, we motivate and justify the use of the greedy variable fixing routine of Lusby et al. [2012] to obtain an integer solution to the original problem.

When implementing column generation, the most conventional approach to obtain an integer solution is to incorporate the procedure in a branch-and-price framework. This is akin to the normal branch-and-bound procedure for solving mixed integer programming (MIP) problems, with the exception that columns are dynamically added at each node of the branch-and-bound tree. Since the relaxed master problem only contains a subset of the possible columns of the original model, the pricing problem(s) must be solved at each node of the branch-and-bound tree to ensure optimality of the procedure.

When solving a standard binary integer program, one typically employs variable branching. With this approach one identifies a fractional variable in the solution to the relaxed problem and then partitions the solution space into two disjoint subspaces (branches). In the *left* branch the variable is bounded downwards (i.e. assumes the value of zero), while in the *right* branch the variable is forced to take its upper bound value of one. This approach, however, does not transfer easily to a column generation setting. In column generation, the property that prevents one from regenerating previously generated variables is the reduced cost of a variable. Existing basic variables have a reduced cost of zero, while existing non-basic variables have a non-negative reduced cost. This is, however, not true for variables with an upper bound. Non-basic variables with a value equal to their upper bound can have a negative reduced cost. One can enforce the bounds on the variables by adding relevant constraints to the master problem. The corresponding dual variables on these constraints can then be included in the reduced cost calculation to ensure that all variables have a non-negative reduced cost. However, it is usually extremely cumbersome to monitor the dual costs for individual variables in the pricing problem. The more preferred approach is a branching strategy that results in minimal changes to the master and pricing problems.

The method of constraint branching developed by Ryan and Foster [1981] is well suited to set partitioning, set packing, and set covering problems, and is often easily incorporated into a column generation approach. This approach requires one to identify two constraints (each with a unit right-hand side) that are covered fractionally in an optimal solution to a relaxed version of the problem. Based on the identification of a such a pair of constraints, one then partitions the solution space into two subspaces. In the left branch the two constraints are required to be covered by the same variable, while in the right branch it is required that they are covered by different variables. From a column generation perspective, it is usually easy to implement such a procedure. Upon identifying a branch, one removes from the current master problem those variables that are inconsistent with branch and then ensures that the subproblem does not return any variables that do not satisfy the branch by modifying the subproblem in some way. Typically this approach is preferred in a column generation context as one can circumvent the need for additional constraints (and the necessary dual

variables) in the master problem.

For the PAS problem, two different types of constraint branches are possible. The first, which could be considered the more conventional one, concerns two different patient-time combinations. One knows that in an optimal integer solution, two different patient-time combinations must be assigned to the same room schedule or to two different ones. Alternatively, one could look at the slightly less traditional constraint branch consisting of a patient-time combination and a particular room type. That is, when branching, one would force a certain patient-time combination to be assigned to a particular room type (or ban it from doing so). However, since the right-hand side of the room type constraint is not required to be unit value, one would expect this branch to be weaker than the traditional constraint branch, where each constraint defining the branch has a unit right-hand side. Indeed, the power of the constraint branch relies on this property. Furthermore, since the number of beds in a particular room type can also be higher than one, forcing a particular patient-time combination to be assigned to a given room type is unlikely to have a dramatic impact on the possibilities for the other patients. Similarly, banning a given patient-time combination from a particular room type is not likely to cause a dramatic change in the objective function value (at least in the early phases of branching). Coupling the constraint branching idea possibilities with a severely degenerate set partitioning problem suggests that an exact branch-and-price strategy is unlikely to be successful. Preliminary tests reinforced this; the fractionality of the solutions did not decrease until a large number of branches had been enforced.

Another, slightly different, approach is to branch on the number rooms of a given room type used. That is, upon solving a relaxed version of the PAS problem, a room type that has a fractional number of its rooms used can be identified and two disjoint subspaces can be identified accordingly. The left branch would ensure that the number of rooms that could be used for the identified room type could be no more than the floor of its current fractional value, while the right branch would ensure that the number of rooms used would be at least the ceiling of its fractional coverage. For example, if we denote the fractional coverage of room type $r \in \mathcal{R}$ in the optimal solution to the relaxed master problem as f_r, one could create the following two branches:

$$\text{Left branch: } \sum_{s \in \mathcal{S}_r} x_{rs} \leq \lfloor f_r \rfloor, \tag{25}$$

$$\text{Right branch: } \sum_{s \in \mathcal{S}_r} x_{rs} \geq \lceil f_r \rceil. \tag{26}$$

The nice feature of this approach is that it can be incorporated into the the column generation framework with no modification to the subproblem structure being necessary. When branching, only the right-hand side (and perhaps the sense) of the relevant room type constraint in the master problem must be changed. It is, however, not a complete branching strategy. That is, a fractional allocation of patient time combinations to the room types might still exist, even though an integer number of rooms of each room type is used. This branching strategy can be extended to incorporate a gender component; i.e. enforce the requirement that a certain number of rooms of a given room type are occupied by one of the genders. It would not be as trivial to implement as simple room type branching, as additional constraints would be needed in the master problem, and the resulting dual variables would also have to be accounted for in the subproblems. Again, however, this is

not a complete branching strategy and would require an additional branching method (i.e. the constraint branch ideas above) to ensure integrality of the final solution.

Using the above branching strategies in a complete branch-and-price framework, it is theoretically possible to find the optimal solution to the PAS problem, albeit potentially very time consuming. Since this is a problem arising in practice, where excessive run times are undesirable, we prefer to implement a heuristic branching strategy where the main focus is to quickly find an integer solution of high quality. The approach we adopt is the aggressive variable fixing strategy of Lusby et al. [2012]. This was proven to be an extremely efficient approach on the generalized set covering problem. The idea is to identify variables to fix in an optimal, fraction solution. Due to the difficulties associated with bounding variables from above in column generation, solely lower bounds are introduced. As the PAS problem is modeled as a binary integer program, enforcing a lower bound of one on a decision variable will be equivalent to fixing the variable at one. In what follows, we describe the variable fixing routine as it applies to the PAS problem.

Consider an optimal solution, x^*, to the relaxed master problem. If $x_{rs} \in \{0,1\}$, $\forall r \in \mathcal{R}, s \in \mathcal{S}_r$, then x^* is also a solution to the original problem, and the algorithm terminates. If this is not the case, we consider the fractional component of each of the variables $f^*_{rs} = x^*_{rs} - \lfloor x^*_{rs} \rfloor$ and determine which one(s) to fix to a new lower bound of one. The criterion for fixing a variable to one is based on a pre-specified threshold τ (with $0 \leq \tau \leq 1$). All variables whose fractional value is at least τ have their lower bounds changed from zero to one. That is, we enforce

$$x_{rs} \geq 1, \forall r \in \mathcal{R}, s \in \mathcal{S}_r. \tag{27}$$

If $f^*_{rs} < \tau$, $\forall r \in \mathcal{R}, s \in \mathcal{S}_r$, we enforce the requirement that the variable with the largest fractional value is fixed to one. The approach effectively generates one path in the full branch-and-price tree. Unlike the work of Lusby et al. [2012], this fixing routine may not result in a feasible integer solution. This is because patients only have a set of room types to which they can be assigned. In other words, not all patients can be assigned to all room types, and some patients are definitely more flexible with respect to this property than others. Through a sequence of potentially greedy variable fixings, it is possible to arrive at a situation where there are no feasible room types left for a particular patient. To guard against this possibility, we offer an alternative strategy. The procedure first identifies for each patient the room schedule that the patient fractionally covers the most (i.e. one room schedule for each patient). These fractional values are first sorted by patient flexibility, where flexibility is measured as the number of remaining feasible room types for the patient, and then in order of decreasing magnitude. Based on this ranking, the largest fractional variable for the room schedule containing the least flexible patient is chosen as the variable to fix. Prior to enforcing the fixing, a look-ahead procedure quickly determines whether fixing the chosen variable will result in infeasibility on the next iteration. If this is the case, we consider the next patient in the ranked list. This has an impact when we change the order in which the patients are ranked. For the decreasing fractional ordering (DFR) we clearly select the column with the largest fractional value regardless of the threshold. If the look-ahead procedure determines that fixing the column will result in infeasibility in the next iteration, then it is not fixed. On the other hand, if it is fixed, then the next fixing has to take into account that the previous column was fixed when checking for infeasibilities. In the increasing flexible ordering case (IFL) we select the columns by increasing flexibility while the column value is still at least equal to the threshold. Again we have to check for infeasibilities, and this may yield another set of columns being fixed compared to the

previous strategy. In case no column above the threshold can be fixed, then the first possible column for each of the orderings are fixed. Note that for the IFL case we may select columns with significantly smaller values than in the DFR case.

An alternative approach of ranking columns is to use a weighted sum of different criteria and choose the column(s) having the smallest weighted sum. An obvious criterion is to use the value of the variable, i.e. $1 - x^*_{rs}$. If we only use this criterion, then we will get a similar approach to the threshold approach described above, with a threshold of 1. Therefore we add another criterion. As discussed above, fixing only on largest variable value may yield a situation where it is not feasible to fix any of the remaining columns in the last iterations. Thus, we take the least flexible patient into account, though in another flavor. On average the penalty of having the patient p in the hospital for a single period in the solution x^* is

$$\overline{d}_p = \sum_{r \in \mathcal{R}} d_{rp} \sum_{s \in \mathcal{S}} \sum_{(p,t) \in \mathcal{W}} \frac{a_{(p,t)s}}{d_p - a_p} x^*_{sr}.$$

For each patient, p, we then say that *good* available compatible room types r are those which are in fact available as well as compatible and also have $d_{rp} \leq \overline{d}_p$. The latter part states that any room type having too high a patient-room penalty will not be considered a good compatible room type. For each room schedule we identify the patient having the least number of good compatible rooms. If we normalize this with the difference between the patient having the largest number of good compatible rooms and the patient having the least number of good compatible rooms then we have the second criterion used in the fixing. We let β^f be the weight of the first criterion (i.e. of $1 - x^*_{rs}$) and β^c be the weight of the second criterion (the normalized number of good available compatible rooms).

Finally, when a lower bound of one is imposed on any given variable, a check is made to see if there is any spare capacity in the room schedule being fixed. If this is the case, those patients not contained in the room schedule, but fractionally assigned to other room schedules, are considered and an attempt is made to assign each of them to the room schedule being fixed without incurring a total penalty greater than what they currently have. If a patient can be assigned (and while there is still capacity in the room schedule), the change is made. Note that if we did not consider this, we would simply waste spare capacity.

We observe that the room schedules in the LP solution tend to be unfilled, which is the reason for filling up room schedules when fixing them. This is likely due to over-capacity of the available beds and rooms, as well as patients may incur the same penalty when being allocated to different room types i.e. the rooms are equally bad from that patient's perspective. As a consequence, it does not matter for the algorithm in which room type the patient is placed, and the column generation will converge towards an arbitrary alternative solution. From the point of view of the variable fixing this may be prohibiting because we have no control of how many different room types are still available. Whenever a room type becomes unavailable, then we risk having patients who cannot have their requirements met. Hence, we are interested in having as many different room types available as possible. To this end, we change the master problem slightly by penalizing the use of the last room of each type. Suppose that δ is the penalty of using the last room of a type, and let $0 \leq v_r \leq 1$ measure the magnitude of usage of the last room. Then we change the objective (1) of the master problem to

$$\min \quad \sum_{r \in \mathcal{R}} \sum_{s \in \mathcal{S}_r} c_{rs} x_{rs} + \sum_{\mathcal{R}} \delta v_r \tag{28}$$

and the upper bound constraint (3) on the number of room types to

$$\sum_{s \in \mathcal{S}_r} x_{rs} \leq N_r - 1 + v_r, \quad \forall r \in \mathcal{R}, \tag{29}$$

leaving the remaining constraints (17) and (5) in the aggregated master problem unchanged. Selecting $\delta > 0$ sufficiently small will force the column generation to choose the alternative solution where as many different room types are available as possible. Note that setting δ too large will most likely increase the objective value in terms of the original objective (1). If we select $\delta = 0$, we will have a model where each solution has the value of the original model.

4 Computational Results

In this section we discuss the effectiveness of the proposed algorithm by comparing its performance on the first six of the 13 benchmark instances provided by Demeester et al. [2010]. The first six instances all contain a time horizon of 14 days, while the remaining seven instances contain between 14 and 91 days and are more difficult. The latter set of instances contain more room properties, different gender properties, and also enforce age constraints on the departments. Due to the increased complexity and size of the model, these have been left as a topic of future research. Furthermore, previous attempts at solving even just the smaller instances using mathematical programming techniques have proven futile. For comparative purposes, we will compare with the best known bounds, running times, and solutions obtained by Ceschia and Schaerf [2011]. In order to make this comparison we use the standard penalties described by Demeester et al. [2010]. As a point of reference, Table 1 provides a summary of the six instances we consider. It reports, the number of rooms (Rooms), the number of room types ($|\mathcal{R}|$), the number of beds (Beds), the number of patients ($|\mathcal{P}|$), the number of time periods ($|\mathcal{T}|$), the number of patient time combinations ($|\mathcal{W}|$), the number of departments at the hospital (Depts.), the number different room properties (Props.), and the number of different specialisms (Specs.).

Instance	Rooms	$\lvert\mathcal{R}\rvert$	Beds	$\lvert\mathcal{P}\rvert$	$\lvert\mathcal{T}\rvert$	$\lvert\mathcal{W}\rvert$	Depts.	Props.	Specs.
1	98	77	286	693	14	2390	4	2	4
2	151	124	465	778	14	3905	6	2	6
3	131	107	395	757	14	3156	5	2	5
4	155	124	471	782	14	3576	6	2	6
5	102	77	325	631	14	2244	4	2	4
6	104	80	313	726	14	2821	4	2	4

Table 1: Problem Instances

Our approach has been implemented in C++ using the MinGW 4.5.2 compiler. We have used the Coin-or interface to the CPLEX 12.2 32bit solver with default settings to solve the LP-relaxation of the master problems, as well as the dual disaggregation problem. The computational tests have been performed on a Windows 7 laptop equipped with an Intel i7 cpu and 8Gb RAM. In the default setting CPLEX uses two parallel processes for solving linear programs, whereas the remaining part of the program is executed on a single core of the cpu.

We begin the analysis with a direct comparison between the lower bound achieved using the proposed methodology, which utilizes dynamic constraint aggregation (DCA), and the current best known lower bounds of Ceschia and Schaerf [2011]. Furthermore, we also compare the DCA with a standard column generation procedure (i.e. no constraint aggregation) and with the DCA approach using the room penalty extension (DCA RP). Table 2 reports the bounds obtained as well as the time taken to achieve them. Note that the objective function value obtained with the DCA RP method must be adjusted to account for the incurred, artificial penalties that we have added. Hence, for this approach there are two objective value columns in the table.

	Ceschia & Schaerf		Std. Col. Gen.		DCA		DCA RP		
Instance	t (s)	LP*	t (s)	LP*	t (s)	LP*	t (s)	LP*	Adj LP
1	13957.8	645.00	292.23	646.933	153.07	646.933	171.16	647.527	646.933
2	46465.5	1111.50	744.13	1120.27	420.33	1120.27	438.08	1121.21	1120.27
3	23574.2	747.00	315.87	758.84	253.56	758.84	273.25	759.731	758.84
4	71291.7	1141.10	1458.64	1144.93	1374.97	1144.93	1604.82	1145.85	1144.93
5	34718.9	620.80	158.34	623.00	102.86	623.00	118.6	623.533	623.00
6	14473.9	787.00	354.30	791.40	168.36	791.40	187.45	792.125	791.40

Table 2: LP Results

	Std. Col. Gen.				DCA				
Instance	It	Rows	Cols	Frac	It	Rows	Cols	Frac	ICols
1	77	2467	12457	623	39	729	10761	584	0
2	111	4029	17817	719	56	879	15366	715	0
3	59	3263	13079	679	43	815	12194	653	9
4	280	3700	23040	670	224	1020	26516	681	474
5	59	2321	9219	542	29	664	8395	540	0
6	76	2901	14781	651	32	765	12785	645	0

Table 3: LP solution statistics

One can observe that for all instances the proposed room schedule model provides a tighter, and in some cases, a much tighter bound than that of Ceschia and Schaerf [2011]. For example, in instance 3 the bound is improved by some 1.6%. Furthermore, in all instances all three column generation based procedures are overwhelmingly faster. Therefore, not only does the model provide tighter bounds, but we can also obtain these bounds in a fraction of the time. The comparison between the standard column generation procedure and that of the DCA approach illustrates the power of the constraint aggregation procedure. With the exception of instance 4, significant gains in the root relaxtion time can be seen. Table 3 provides some summary statistics for the standard column generation procedure and the DCA approach. It reports the number of iterations (It), the number of rows and columns in the master problem, and the number of fractional columns at the optimal LP solution. For the DCA approach, it also reports the number of incompatible columns identified at the last iteration of the column generation (ICOL). One can see that the increased speed up from DCA most likely comes from the reduced number of iterations and a smaller master problem. The small difference in solution times between the two different approaches for instance 4 is probably due to the large number of incompatible columns; a reaggregation of

$\mathcal{W}$ into a new set of clusters is required every time we wish to force incompatible columns into the master problem

Finally, we present the results obtained using the greedy variable fixing routine described in Section 3.4 to force integrality of the solutions. The results of these tests are shown in Table 4 and table 5. Both of these tables include the best known solution values (CS UB) reported by Ceschia and Schaerf [2011], which are compared to our results. Our best solutions are underlined in the tables. If our results are equal to those reported by Ceschia and Schaerf [2011], then the result is shown in italics. Results shown in boldface are where our method achieves solutions that are better than the best solutions reported in the literature.

We test and compare two different approaches: the first fixes just one variable (room schedule) per iteration, while the second fixes all variables whose value is above the threshold τ, where $\tau = 0.8$. For each of the two methods we further test and compare two different strategies. In the single variable case we first look at the impact of setting the room penalty to zero ($\delta = 0$) or 0.01 ($\delta = 0.01$). That is, in the latter approach one incurs a penalty of 0.01 for using the last room of any type. We also consider the strategy in which one considers a weighted sum fractionality of the room schedules (see Section 3.4) and the normalized number of good available compatible rooms. Here we test two alternative strategies, one where we only use the the largest fractionality of column ($\beta^f = 1, \beta^c = 0$) and a second where the two criteria are weighted evenly ($\beta^f = \beta^c = 1$). Table 4 summarizes the results. The table reports, for each fixing strategy, the time used to obtain the solution and its value (UB). Recall that the fixing strategy will identify one solution only and that it is heuristic in nature.

		$\beta^f = 1.0$, $\beta^c = 0.0$				$\beta^f = 1.0$, $\beta^c = 1.0$			
		$\delta = 0$		$\delta = 0.01$		$\delta = 0$		$\delta = 0.01$	
Instance	CS UB	t(s)	UB	t(s)	UB	t(s)	UB	t(s)	UB
1	655.6	489.98	**655.2**	666.47	657.2	508.14	695.8	685.15	686.6
2	1137.2	1270.90	1160.4	1330.49	1157.4	1307.08	1149.8	1298.98	**1132.8**
3	773.6	709.67	**768.2**	920.68	792.6	666.35	**769.2**	963.25	777.6
4	1172.2	5628.51	1210.0	6226.98	1247.6	5678.29	1259.2	5461.72	1192.6
5	625.6	300.67	628.0	401.59	629.6	297.43	634.0	360.55	**624.8**
6	798.0	637.27	**792.6**	672.13	798.0	1003.45	846.2	724.54	810.2

Table 4: Variable fixing using weights

From the table, one can observe that the solutions obtained using the $\delta = 0$ strategy tend to outperform the $\delta = 1$ approach. The $\delta = 0$ setting with $\beta^c = 0$ provides three new best known solution values (test instances 1, 3, and 6), while the others are not far from their respective best known solutions. With the exception of instance 4, all solutions are found within 20 minutes. The results further suggest that using the $\beta^f = \beta^c = 1$ strategy is beneficial only if the the room penalty approach is also used. This setting ($\delta = 0$,$\beta^f = \beta^c = 1$) also produces two solutions (test instances 2 and 5) which are better than those previously reported.

In comparison, Table 5 reports the results of fixing all variables in a fractional LP optimal solution with value at least 0.8. Here we also test and compare two different strategies, namely the decreasing fractional ordering (DFL) and the increasing flexible ordering (IFL). For each of these we also test the impact of setting $\delta = 0$ and $\delta = 0.01$. From the results one can see that, due to the fact that we have the possibility of fixing more variables per iteration, the solution times decrease in comparison to the single variable fixing case. However, we also

		Decreasing fractional order (DFR)				Increasing flexible ordering (IFL)			
		$\delta = 0$		$\delta = 0.01$		$\delta = 0$		$\delta = 0.01$	
Instance	CS UB	t(s)	UB	t(s)	UB	t(s)	UB	t(s)	UB
1	655.6	458.75	681.2	952.74	666.4	379.37	*655.6*	710.55	657.6
2	1137.2	1147.55	1176.2	1416.87	1184.8	1259.75	1150.8	1224.09	1146.8
3	773.6	572.33	-	745.32	781.0	571.23	787.0	852.33	796.2
4	1172.2	4484.55	1251.2	5347.90	1179.0	5097.36	1188.6	6197.94	1213.6
5	625.6	281.20	*625.6*	428.31	628.8	252.53	**624.0**	367.26	626.4
6	798.0	1026.09	-	734.60	803.8	638.27	837.4	723.10	806.2

Table 5: Variable fixing using threshold 0.8

increase the possibility of not finding a feasible solution and in some cases we do not (see for example instances 3 and 6 with no room penalty and the DFL approach). The remaining three strategies seem equally good, as not one strategy dominates the others. Including the last room penalty is effective in the sense that we can produce a feasible solution to all instances. However, not including the room penalty and considering patients in terms of flexibility also has potential. This approach yields the best known solution for instance 5.

In general, it should be noted that the best feasible solutions are very close to the lower bound that we have found. For example, in test instance 5 we have a gap of only 1 unit and in instance 6 we have a gap of 1.2 units. The largest gap on the newly found best solutions is 1.28% above the lower bound for test instance 1. This indicates that the lower bound is very tight in many cases. For the best known solution for instance 4 found by Ceschia and Schaerf [2011] the gap is 2.38%, which is still small, while our best solution for instance 4 has a gap of 2.98%, which is still reasonable.

The computational results show that a column generation based heuristic approach using aggressive variable fixing is a viable approach for identifying high-quality solutions for PAS. While in most cases providing feasible solutions, the column generation based heuristic also provides a quality measure through the lower bound obtained in the root node. This is, in our opinion, a significant advantage compared to heuristics in general.

5 Conclusion

The Patient Admission Scheduling problem is an intriguing and important part of hospital management. For hospitals with a central planning unit, producing an optimal or near-optimal plan can be highly important for patients as well as the hospital – a good plan will result in better care with a better utilization of resources and a focus on the right resources. It will minimize extra work in moving patients around the hospital and allocating the patients to the wards best suited for their needs given the overall set of elective patients.

In this paper we have developed a new state-of-the-art optimization-based approach for the PAS problem as defined by Demeester et al. [2010]. The method is based on column generation to give tight bounds and a branch-and-bound framework – leading to a branch and price set-up – to guide the solution process in finding near-optimal solutions. The framework is based on variable fixing. In order to efficiently handle the relatively large master problem of the method, dynamic constraint aggregation has been incorporated in the approach. This reduces the master problem significantly although, due to the administrative overhead, the reduction is not fully carried over into a running time reduction.

On the six first instances from Demeester et al. [2010] we produce new best solutions in five cases. Overall, it should be noted that the gaps we obtain are in general small, for

example the largest gap on the newly found best solutions is 1.28% above the lower bound.

In conclusion, our optimization-based heuristic is currently the best approach for the small instances of the PAS problem. For the larger instances the approach needs further refinements. For future research we could utilize the fact that even in a long time horizon most almost all patients are usually only in the hospital for a short while. This suggests that a rolling-horizon time window approach would be able to give us near-optimal solutions even for larger instances than the current instances over 14 days. Another remaining challenge is also to be able to produce the best solutions with one parameter setting or incorporate a self-adjusting approach on the parameters in the method.

References

B. Bilgin, P. Demeester, M. Misir, W. Vancroonenburg, and G. Vanden Berghe. One hyper-heuristic approach to two timetabling problems in health care. *Journal of Heuristics*, 18(3):401–434, 2012.

S. Ceschia and A. Schaerf. Local search and lower bounds for the patient admission scheduling problem. *Computers and Operations Research*, 38:1452 – 1463, 2011.

N. Chen, Z. Zhan, J. Zhang, J. Zhang, O. Liu, and H. Liu. A genetic algorithm for the optimization of admission scheduling strategy in hospitals. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC) 2010*, pages 1 – 5. IEEE, 2010.

P. Demeester, W. Souffriau, P. De Causmaecker, and G. Vanden Berghe. A hybrid tabu search algorithm for automatically assigning patients to beds. *Artificial Intelligence in Medicine*, 48:61–70, 2010.

J. Desrosiers, M. Lübbecke, and M. M. Solomon. Column generation. In Guy Desaulniers, Jacques Desrosiers, and Marius M. Solomon, editors, *A Primer in Column Generation*, chapter 1. Springer: New York, 2005.

I. Elhallaoui, D. Villeneuve, F. Soumis, and G. Desaulniers. Dynamic aggregation of set-partitioning constraints in column generation. *Operations Research*, 53(4):632 – 645, 2005.

I. Elhallaoui, A. Metrane, F. Soumis, and G. Desaulniers. Multi-phase dynamic constraint aggregation for set partitioning type problems. *Mathematical Programming Series A*, 123:345–370, 2010.

I. Elhallaoui, A. Metrane, G. Desaulniers, and F. Soumis. An improved primal simplex algorithm for degenerate linear programs. *Informs Journal on Computing*, 23(4):569–577, 2011.

F. Guerriero and R. Guido. Operational research in the management of the operating theatre: a survey. *Health Care Management Science*, 14(1):89–114, 2011. ISSN 13869620, 15729389. doi: 10.1007/s10729-010-9143-6.

P. R. Harper. A framework for operational modelling of hospital resources. *Health Care Management Science*, 5: 165 – 173, 2002.

A. K. Hutzschenreuter, P. A. N. Bosman, I. Blonk-Altena, J. Aarle, and H. L. Poutré. Agent-based patient admission scheduling in hospitals. In *Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008) – Industry and Applications Track*, pages 42 – 52, May 2008.

P. Jittamai and T. Kangwansura. A hospital admission planning model for emergency and elective patients under stochastic resource requirements and no-shows. *2011 IEEE Internationl Conference on Industrial Engineering and Engineering Management (IEEM)*, pages 166–170, 2011. ISSN 21573611, 2157362x. doi: 10.1109/IEEM.2011.6117900.

R. J. Kusters and P. M. A. Groot. Modelling resource availability in general hospitals design and implementation of a decision support model. *European Journal of Operational Research*, 88:428 – 445, 1996.

R. M. Lusby, A Dohn, T. M. Range, and J. Larsen. A column generation based heuristic for rostering with work patterns. *Journal of the Operational Research Society*, 63:261 – 277, 2012.

T. M. Range, R. M. Lusby, and J. Larsen. Solving the selective multi-category parallel-servicing problem. Working paper, 2012.

D. M. Ryan and B. A. Foster. An integer programming approach to scheduling. In A. Wren, editor, *Computer Scheduling of Public Transport*, pages 269 – 280. North-Holland Publishing Company, 1981.

D. Villeneuve. *Logiciel de génération de colonnes*. PhD thesis, Université de Montréal, Montreal, Quebec, Canada, 1999.