Annals of Operations Research 115, 207–225, 2002

Crew Assignment via Constraint Programming: Integrating Column Generation and Heuristic Tree Search [*]

MEINOLF SELLMANN sello@uni-paderborn.de
University of Paderborn, Department of Mathematics and Computer Science, Fürstenallee 11, D-33102 Paderborn, Germany

KYRIAKOS ZERVOUDAKIS and PANAGIOTIS STAMATOPOULOS {quasi,takis}@di.uoa.gr
University of Athens, Department of Informatics and Telecommunications, Panepistimiopolis, 157 84 Athens, Greece

TORSTEN FAHLE tef@uni-paderborn.de
University of Paderborn, Department of Mathematics and Computer Science, Fürstenallee 11, D-33102 Paderborn, Germany

Abstract. The *Airline Crew Assignment Problem (ACA)* consists of assigning lines of work to a set of crew members such that a set of activities is partitioned and the costs for that assignment are minimized. Especially for European airline companies, complex constraints defining the feasibility of a line of work have to be respected. We developed two different algorithms to tackle the large scale optimization problem of Airline Crew Assignment. The first is an application of the Constraint Programming (CP) based Column Generation Framework. The second approach performs a CP based heuristic tree search. We present how both algorithms can be coupled to overcome their inherent weaknesses by integrating methods from Constraint Programming and Operations Research. Numerical results show the superiority of the hybrid algorithm in comparison to CP based tree search and column generation alone.

Keywords: airline crew assignment, hybrid OR–CP method, constraint programming based column generation, set partitioning

1. Introduction

Scheduling flying crews of airline companies is a hard combinatorial problem, given the complexity of the constraints that have to be satisfied and the huge search space that has to be explored. The problem is often tackled by breaking it down into the crew pairing and the crew rostering (or assignment) subproblem. In the crew pairing part,

[*] This work was partly supported by the PARROT Project, partially funded by the ESPRIT Programme of the Commission of the European Union as project number 24 960, by the German Science Foundation (DFG) project SFB-376, and by the German Ministry for Education and Research Bmb+f (PARPAP project 01 HR 9955).

basic activities such as *flight legs* (flights without stopover) are grouped into *pairings*. The latter ones are lines of work for one or more days starting and ending at a home base. Then, in the crew assignment phase, these pairings are assigned to crew members.

Although easier than the original problem, both subproblems are still hard to solve. Generally, Operations Research (OR) and Constraint Programming (CP) techniques are available to solve both problems, since they have drawn the interest of both scientific communities for many years until today. Most industrial software is based on OR techniques. Especially for European airlines though, there are very strict rules enforced by legislation, unions, etc. that define the feasibility of schedules. Thus, because a huge amount of computational effort is put into the generation of infeasible lines of work, common OR-based generate and test approaches are not efficient enough. We show how Constraint Programming can be incorporated to overcome typical weaknesses of OR approaches. For a recent overview on optimization problems and solution techniques in the airline industry, we refer to [12,19].

By construction, OR methods view a problem globally, taking into account all variables and usually more than one or even most constraints at a time. By calculating upper and lower bounds on the costs, they show a good ability to identify promising parts of the search space. However, they often suffer from minor local conflicts, which might prevent a feasible solution from being found. On the other hand, CP methods can efficiently handle feasibility problems by resolving local conflicts using algorithms based on arc consistency and advanced search techniques. Respectively, CP methods lack the ability to view the variables and constraints of a problem globally. Therefore, they often have problems when stuck in local optima.

During the last decade, some work was done on the crew rostering problem. Column generation methods have proved to be quite successful – see, e.g., [4,6,13]. For solving the railway crew rostering problem, which is similar, but not identical to the airline crew rostering problem, Caprara et al. developed both an OR and a CP based approach – see [2,3]. For the latter one, a lower bound from the OR field was used to improve on the efficiency.

Within the PARROT ESPRIT Project 24 960, we developed two different approaches to tackle the Airline Crew Assignment Problem (ACA): a CP based heuristic tree search approach (HTS) [17], and one following the CP based Column Generation Framework (CGA) [5,11]. We show how these two approaches can be combined to overcome their inherent limitations. A preliminary version of this work has appeared in [15].

The paper is structured as follows. In section 2, we define the ACA Problem and give the characteristics of the airline test cases used for the experiments. In section 3, we discuss two autonomous approaches to solve the ACA. Detailed ways of how these approaches can be integrated to develop an efficient hybrid algorithm for tackling the ACA are presented in section 4. Finally, in section 5, numerical results show the superiority of the hybrid algorithm compared to the individual approaches.

2. The Airline Crew Assignment Problem

Given a set of crew members, a set of pairings, a set of rules and a cost function, a *roster* is an assignment of a subset of pairings to one specific crew member. A *schedule* is a set of rosters such that all rules are obeyed and every pairing is assigned to exactly one crew member. Rules may concern a *single crew member* or *multiple crew members*. Single crew member rules regard each individual crew member's roster, stating for example that no two temporally overlapping pairings can be assigned to the same crew member. Multiple crew member rules aim at more than one crew member, stating for example that two given pairings must be assigned to two crew members out of which at least one must have a certain level of experience. The cost function associates a cost with every legal schedule, and its minimization is desired.

In our case, every rule in the rule set only deals with just one single crew member, and the objective function is linear over the rosters. That means that only single crew member rules can be modeled and that the cost of the entire solution to the ACA is defined as the sum of the costs of the selected rosters. More formally:

2.1. Definition

Let $k, m, n \in \mathbb{N}$ and let $C := \{1, \ldots, m\}$ the *set of crew members* and $T := \{1, \ldots, n\}$ the *set of pairings*.

1. Let $R := C \times 2^T$. Every $r \in R$ is called a *roster* and R is called the *set of all possible rosters*.
2. Let $B := \{0, 1\}$ and $H := \{h_1, \ldots, h_k \mid h_i : R \to B \; \forall\, 1 \leqslant i \leqslant k\}$. Every $h \in H$ is called a *(single crew member) rule* and H is called a *rule set*.
3. A roster $r \in R$ is called *legal (with respect to a rule set H)* iff $h(r) = 1 \; \forall\, h \in H$. $L(H) := \{r \in R; r \text{ is legal}\}$ is the *set of legal rosters (with respect to the rule set H)*.
4. $f : R \to \mathbb{Q}^+$ is called a *cost function*.
5. The *Crew Assignment Problem (ACA)* is to minimize $\sum_{1 \leqslant i \leqslant m} f((c_i, t_i))$, where $(c_i, t_i) \in L(H) \; \forall\, 1 \leqslant i \leqslant m$ s.t.

   (a) $\{c_1, \ldots, c_m\} = C$,

   (b) $\bigcup_{1 \leqslant i \leqslant m} t_i = T$ where $t_i \cap t_j \neq \emptyset \Rightarrow i = j \; \forall\, 1 \leqslant i, j \leqslant m$.

The model as stated above neither allows non-linear objectives when combining rosters, nor permits to restrict the combination of rosters by additional multiple crew member rules one might be interested in when tackling real life applications. Nevertheless, both methods we present to solve the above problem allow to treat linear multiple crew member rules as well.

2.2. The airline test cases

We consider test cases stemming from two European airline companies. The instances of company A consist of 50–65 crew members and 766–959 pairings. Company B has 7–30 crew members and 129–279 pairings. Case A covers a planning period of one calendar month, while data sets for B cover two weeks. While case B incorporates mainly 1–2 day pairings, A considers pairings of duration less than 24 hours.

The objective of company B is to achieve a fair distribution of activities over all crew members, whereas in A we aim at satisfying as many preferences expressed by the crew members as possible by minimizing dissatisfaction.

Importantly, the rule sets in both cases are distinct. In A, typical rules such as succession rules and rest time rules, but also more complicated ones like rules ensuring a minimum of days off within gliding windows of variable lengths are incorporated. Also, rules guaranteeing minimum and maximum flight time are enforced. All rules in A are hard constraints, meaning that if they are violated, the solution is considered infeasible.

In B, we consider flight time rules that limit the time actually flown by the crew within certain time periods. These rules are also strict.

The main difference between the two test cases regarding the algorithms we developed is due to the fact that company B does not insist on a partitioning of the work, i.e. in that test case restriction 5(b) is relaxed to $\bigcup_{1\leqslant i\leqslant m} t_i \subseteq T$. Obviously, this difference requires that our algorithm is able to incorporate two different types of master problems.

3. Two approaches to solve the ACA

In this section, we introduce two approaches for the ACA that we want to combine later. As our main focus in this paper is on integration, for further insights we refer to [5,11,17] describing the algorithms in more detail.

One goal within the PARROT project was the development of generic tools that are able to treat different rules and regulations that typically arise in airline companies. Particularly for European airlines, these rules are very complex and often non-linear. It was therefore decided to model the rules and regulations as a constraint program. Hence, both approaches and the resulting integrated approach are based on a CP core.

3.1. CP based column generation approach

The definition of the ACA as stated above allows to apply the column generation principle, i.e. it can naturally be decomposed into the *subproblem* of generating legal rosters and the set partitioning *master problem*. The latter one is an integer program (IP) that ensures 5(a) and 5(b) of definition 2.1:

$$\min \sum_{i=1,\dots,k} f\big((c_{\varphi(i)}, t_i)\big)x_i$$

$$\text{s.t.} \quad \sum_{\substack{i=1,\ldots,k \\ \text{and } \varphi(i)=j}} x_i = 1, \qquad j = 1, \ldots, m, \tag{1}$$

$$\sum_{\substack{i=1,\ldots,k \\ \text{and } s \text{ belongs to } t_i}} x_i = 1, \quad s = 1, \ldots, n, \tag{2}$$

$$x_i \in \{0, 1\}$$

where $\varphi : \{1, \ldots, k\} \to \{1, \ldots, m\}$ maps a column number to a crew member. The m constraints in (1) assign exactly one line of work to each crew member. The n constraints in (2) ensure that all activities are covered exactly once. In this model, every (legal) roster corresponds to a 0–1 column.

The subproblem consists of finding rosters respecting all rules and improving the objective. From linear programming (LP) duality theory, it is known that columns with negative reduced costs are candidates for such an improvement. Notice that duality theory is only valid for the LP-relaxation of the original IP. Thus, pure column generation must be viewed as a heuristic only. To prove optimality of the IP model, column generation has to be extended to a branch and price approach [1].

We first generate a bunch of individual lines of work and then try to combine them to partition the entire work. When solving the LP-relaxation of the master problem, we get dual information that allows to search for potentially improving columns. That is, in the subproblem we try to generate new rosters that have negative reduced costs. Those rosters are added to the master problem, which is solved again, and so on until no more rosters with negative reduced costs can be computed or until a certain iteration limit is reached.

Selecting an optimal set of non-overlapping activities respecting the rule set can be interpreted as the problem of finding a constrained shortest path in a weighted directed acyclic graph (DAG) G (see figure 1). There, consistency checking allows to remove arcs and nodes from G that could cause the construction of illegal or non-improving rosters.

Due to complex and possibly non-linear single crew member rules, generating legal rosters with negative reduced costs can be very difficult. Moreover, rule sets vary

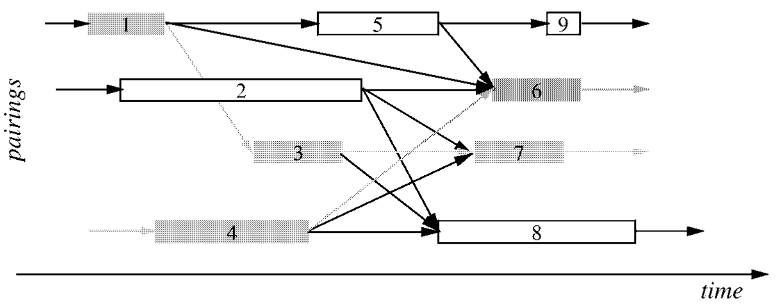

Figure 1. Constructing an optimal and legal roster is equivalent to finding a *constrained shortest path* in a weighted DAG.

from airline to airline and have no common structure that could easily be exploited to design a generic efficient constrained shortest path algorithm that can cope with any rule set. Therefore, we apply a CP search to generate legal rosters. As we are only searching for individual lines of work with associated negative reduced costs, we incorporate an optimization constraint that performs reduced cost propagation. That is, instead of searching for constrained shortest paths, we rather introduce a *shortest path constraint*.

This framework has been formalized by Junker et al. [11] and is called CP based column generation approach. It was applied in [5] and has proven to be generic and yet efficient for the ACA. However, it suffers from certain drawbacks.

3.1.1. Set partitioning – set covering

The major obstacle for CGA is the set partitioning (SPP) structure of the master problem. Finding a feasible solution to the SPP is NP-hard already [7]. Moreover, the dual information gained from equation constraints is more difficult to exploit than that of cover or packing constraints. Therefore, we would like to relax the master problem to a set covering formulation (that remains an NP-hard problem but can be solved much more easily in our case) by only requiring the pairings to be flown by one *or more* crew members, i.e., we relax (2) to $\sum_i x_i \geqslant 1$. Then, however, to compute a legal schedule, we must decide which crew member finally gets an overcovered pairing assigned.

3.1.2. Feasible solutions for set partitioning

To obtain a formulation that guarantees that we can always find a feasible solution, we add two types of *dummy columns*: The first type of columns covers exactly crew member i, the second exactly one activity j, for all $i = 1, \dots, m$ and $j = 1, \dots, n$. That is, we allow empty rosters and unassigned activities. By setting the costs for choosing a dummy column to an arbitrary high value, we make sure that they only become part of an optimal solution if the original master problem was infeasible.

Although this procedure works, to achieve meaningful dual information of the master problem, the solution should not be spoiled by dummy costs. Thus, it would be better to generate an initial set of rosters that contains an entire work partitioning schedule.

3.2. Heuristic tree search CP approach

The other algorithm developed to tackle the ACA is the heuristic tree search CP approach. In that HTS, each complete feasible solution of the ACA is constructed by solving the corresponding constraint satisfaction problem – see [17]. The problem is modeled by a set of variables, which correspond to assignable pairings. For each pairing, there is a variable the domain of which represents the crew members that can possibly be assigned to the pairing.[1] For each constrained variable representing the assignment of a

[1] It is assumed that every pairing can only be assigned to one crew member. In case there are more than one crew members necessary to staff a pairing, copies of the pairing are created, and each copy can again only be assigned to a single crew member.

pairing, its initial domain comprises all available crew members. The posting of the appropriate constraints reduces the domains of these variables by removing crew members that cannot be allocated to the corresponding pairings. This is possible, for example, due to preassigned activities, regulation violations because of the crew member's history, etc. The search tree of the problem is created by iterating over pairings in some specific way and assigning each pairing to a crew member.

Each level of the tree corresponds to the assignment of a pairing. The branch followed from a node represents the allocated crew member to the pairing. Each non-leaf node corresponds to a partial assignment, identified by the path from the root to the node. Leaf nodes correspond to complete legal assignments, i.e. (not necessarily optimal) feasible solutions of the problem. Each allocation of a crew member to a pairing activates the constraint propagation mechanism. More branches of the tree are pruned, as values which are inconsistent with the posted constraints are removed from variables' domains. For example, the assignment of a pairing to a crew member causes the removal from the domain of the crew member of all other pairings that overlap with the one just assigned. When a node is proved to be a dead-end, which means that one or more pairings cannot be assigned to any crew member, backtracking occurs, and decisions taken before are reconsidered.

The constraints of the problem are the regulations of the airline at hand that dictate which rosters are acceptable and which are in violation of the airline rules. A solution to a constraint satisfaction problem is any assignment of values to variables that respects all constraints. A feasible solution to the ACA, formulated as a constraint satisfaction problem, is any assignment of crew members to pairings such that all airline rules and regulations are respected. Then, the objective function is optimized by searching for improving solutions only.

Regarding the way the search tree is traversed, we developed a variety of search methods. Essentially, the distinctions between search methods emanate from the fact that they set choice points at different nodes.

3.2.1. Tree traversal

A variety of search methods for traversing the problem tree exists in the literature. The oldest, most popular and, by far, most widely used search method is Depth First Search (DFS). The main drawback of DFS is that, even for problems of moderate size, it only explores a very small portion of the search tree at the lower left.[2] Although we implemented and tested DFS, we found that it does not perform well for our problem.

Innovation in the field came from the notion of *discrepancy*. At a given node, a heuristic function provides an estimate of which branch the search should follow, as the one that is most likely to contain solutions (or solutions of good quality in the case of optimization). Always following the heuristic advice defines a unique path that is said to contain no discrepancies. Following the heuristic advice except for one case defines

[2] It is common practice to regard the branches under a node as ordered according to a heuristic function. Following the advice of a heuristic means to go "left" down the search tree.

paths of discrepancy one. A path contains a discrepancy whenever the heuristic advice is not followed.

Limited Discrepancy Search (LDS) [8] is an iterative search method. In the ith iteration, it explores all paths with i or less discrepancies. In the ith iteration of the original LDS method, paths with discrepancies higher up the tree are explored before the ones where the discrepancies occur further to the bottom. The intuitive justification for that approach is that a heuristic is more likely to be wrong higher up the tree, where information is limited. We implemented a variant of LDS. Our variant searches paths with discrepancies lower down the tree before ones with discrepancies higher. Its advantage is that time consuming descends from near the root towards the leaves are avoided. Also, our variant is not iterative. It searches those paths having i or less discrepancies and then exits. Thus, it is not complete. Practically, however, the parameter i can be chosen so that a big enough portion of the tree is explored. In our experiments, this portion of the tree was much bigger than a modern computer could explore in a reasonable amount of time. We call this variant *modified Exact Discrepancy Search* (mEDS).

Depth-Bounded Discrepancy Search (DDS) [18] is also an iterative method. In the ith iteration, it explores all paths where discrepancies occur before depth i. In contrast to LDS, a path with many discrepancies high in the tree is explored before a path with very few discrepancies low in the tree. This is also justified by the assumption that heuristics tend to fail with a higher probability on top of the tree.

Large Neighborhood Search (LNS), introduced in [16], incorporates local search techniques within the CP framework. The idea is to restrict the search within a fragment of the problem search space. In this way, minor local improvements can be made, which would go unnoticed by most search methods. A reduced search space for a problem with a set of unknown variables V and a known feasible assignment $\mathcal{A}$ can be created as follows: A large subset V_1 of V is selected. All assignments in $\mathcal{A}$ for variables in V are fixed and thus a partial solution is created. Search is performed in the remaining variables with any of the above search methods. After this search is finished (either because the search subspace has been exhausted or any other termination criterion is met), another subspace is selected and the process is repeated. The advantage of LNS is that local improvements are discovered easily, and the objective value is improved quickly. The disadvantage is that the search space cannot be viewed globally. Thus, it is likely that important improvements are missed. A rational strategy when using LNS is to use one of the search methods above in the beginning to guide the search towards a promising area of the search space and to use LNS afterwards to resolve minor local conflicts.

4. Integration

We present two ways of integrating both methods each one motivated by different problem cases. In the first problem case, the construction of a feasible schedule is difficult due to very strict rules called for by the airline company.

We observe that CGA eventually gets close to solutions of good quality, but minor inconsistencies delay it disproportionately long. We show that this can be overcome effectively by letting the CGA approach solve a relaxed (that is Set Covering) version of the problem and then handing possibly overcovered (and thus infeasible) solutions to the HTS approach for fixing.

In the second problem case, the rule set is not that strict. The CGA approach alone proceeds as expected. However, the initial time spent for driving dummy columns out of the basis is considerable. In this phase, dual values are not very meaningful, because penalties dominate the objective. We show how the HTS method can help attacking the problem.

The issue that arises is that of the general applicability of each hybrid method and the possibility of them being combined to one single meta-hybrid, which would be generally applicable. We address these issues in the end of section 5.

4.1. First way of integration: transforming a Set Covering into a Set Partitioning solution

The first method is applied on case A. In this company, no pairing can be left unassigned. Moreover, there is a relatively large number of pairings with respect to the number of crew members (for example 959 pairings/65 crews on a typical monthly problem). These conditions make finding a feasible solution difficult for the CGA approach. On the other hand, the HTS approach is able to construct feasible solutions by using sophisticated search methods and heuristics tailored for the specific problem. However, after a short while no improving solutions can be found.

Algorithm 1. Top level algorithm for the first method

```
1. $\mathcal{A}_{HTS} \leftarrow$ HTSOptimize($V$, DefaultSVar, DefaultSVal)
2. repeat
3.    $\mathcal{A}_{CGA} \leftarrow$ CGAOptimize
4.    $(V_1, V_2, V_3) \leftarrow$ Partition($\mathcal{A}_{HTS}$, $\mathcal{A}_{CGA}$)
5.    for all $v \in V_3$ do
6.       $v \leftarrow a(\mathcal{A}_{CGA}, v)$
7.    $\mathcal{A}_{HTS} \leftarrow$ HTSOptimize ($V_1 \cup V_2$, RepairSVar($V_1 \cup V_2$, $\mathcal{A}_{CGA}$, $V_2$),
         RepairSVal($V_1 \cup V_2$, $\mathcal{A}_{CGA}$, $V_2$))
8.    $\mathcal{A}_{HTS} \leftarrow$ LnsOptimize($V$, RepairSVar($V$, $\mathcal{A}_{CGA}$, $V_1 \cup V_2 \cup V_3$),
         RepairSVal($V$, $\mathcal{A}_{CGA}$, $V_1 \cup V_2 \cup V_3$), $\mathcal{A}_{HTS}$)
9. until stopping condition
```

We overcome the problems of both methods by letting the CGA approach find *Set Covering* instead of Set Partitioning solutions. That is, we relax the pairing partitioning constraints (2) by only requiring that every pairing is assigned to *at least* one crew member. The columns generated by the CGA approach are much more easily combinable to SCP solutions. Then, the conversion of SCP to SPP solutions is assigned to the HTS approach, which can resolve local conflicts efficiently by using sophisticated propagation algorithms. An outline of the procedure is shown in algorithm 1. Here, V is the set of all variables, $\mathcal{A}_X$ is a tuple of assignments $\langle v, x_v\rangle$ of values x_v to variables v generated by approach X, $a(\mathcal{A}, v)$ is a function which returns the value of variable v in assignment $\mathcal{A}$, DEFAULTSVAR and DEFAULTSVAL are respectively the variable and value selection functions normally used by the HTS approach, REPAIRSVAR and REPAIRSVAL are the corresponding heuristics used for repairing Set Covering solutions, and HTSOPTIMIZE and CGAOPTIMIZE are the HTS and CGA optimization functions. PARTITION is a function which will be explained shortly. LNSOPTIMIZE performs optimization using the LNS method. The time span of the entire schedule is divided into successive time windows. All activities within such a window form a search subspace.

We now explain this algorithm in greater detail. In the first line, one or more initial solutions are found by the HTS approach. This initialization step provides the algorithm with a set of columns, which can be combined to feasible solutions. Not much time is devoted to this phase. The variable and value selection heuristics that would normally be used by the HTS approach are applied here. Any of the methods presented in the previous sections can be plugged in. However, we found mEDS to perform best in our case. The columns constituting these solutions are handed to the CGA approach for optimization in line 3. The solution produced in this step is correct except for the fact that some pairings are assigned to more than one crew member, which is not legal.

The next task is to use the information found in $\mathcal{A}_{CGA}$ to construct a feasible solution. Let V_1 be the set of variables which correspond to overcovered pairings. One optimistic approach would be to assign the values of the assignment $\mathcal{A}_{CGA}$ to all the variables in $V \setminus V_1$ and let the HTS approach perform a search in the space of the variables in V_1. This, however, could lead to a failure, since it is not known that the partial solution obtained is extendible to a feasible solution.

For some scheduling problems, such as the vehicle routing problem with time windows (see, e.g., [14]), a partial solution can easily be extended by removing entries for overcovered rows from all but one of the corresponding columns. In our case, though, this approach is not generic enough, as certain rules may cause the resulting rosters to be infeasible. For example, a minimum flight time rule might be violated if a pairing is removed from an otherwise feasible roster. We say that such a rule destroys the *legal subroster property* of a rule set.

We can distinguish three subsets of variables in V: The set V_1 that consists of variables that correspond to overcovered pairings in $\mathcal{A}_{CGA}$, the set V_2 that consists of variables which have different values in $\mathcal{A}_{CGA}$ and $\mathcal{A}_{HTS}$, and the set V_3 which corresponds to variables having the same value in both assignments.

Function PARTITION partitions V in exactly this manner. Assignments of variables in V_3 are known to be extendible to a full solution, since one has already been found. Thus, because there is no information which suggests the contrary, they are realized as soon as possible in each iteration. Assignments in set V_2 may be considered as almost certain. However, they should be realized in a manner that allows backtracking. These issues are handled in line 7 with respect to the search method and the heuristics used.

CGA does not provide meaningful information for variables in V_1, so HTS performs the search for assignments to these variables using the default heuristics.

The variable and value selection functions are modified as shown in algorithm 2. There, the variables that will be taken into account are variables in S. V is a subset of S for which assignments exist in $\mathcal{A}$. For example, when the variable selection rule is invoked in line 7 of algorithm 1, S is $V_1 \cup V_2$, V is V_2 and $\mathcal{A}$ is $\mathcal{A}_{CGA}$. In this case, the variable to be assigned next is any variable in V_2 for which its suggested value exists in its domain. In other words, all possible assignments in $\mathcal{A}_{CGA}$ are realized as soon as possible, in accordance to the intuitive belief that they would most probably lead to an area containing improving solutions. If this is not possible, then a variable in V_1 is selected, and the default heuristic is used.

Algorithm 2. Heuristics for the first method

```
REPAIRSVAR($S$, $\mathcal{A}$, $V$)
1. $v \leftarrow$ NIL
2. **for all** unbound variables $v \in V$ **do**
3.     **if** $a(\mathcal{A}, v) \in D_v$ **then**
4.         **return** $v$
5. **return** DEFAULTSVAR($S$)
REPAIRSVAL($S$, $\mathcal{A}$, $V$, $v$)
1. **if** $v \in V$ and $a(\mathcal{A}, v) \in D_v$ **then**
2.     **return** $a(\mathcal{A}, v)$
3. **else**
4.     **return** DEFAULTSVAL($S$, $v$)
```

Whenever possible, the value selection heuristic assigns the value suggested by CGA to each variable. Two important details are worth noting:

1. The variable selection heuristic is consulted *every time* a new assignment has to be made in the HTS search. That is, if a variable v is selected (because $a(\mathcal{A}_{CGA}, v) \in D_v$) and then, for any reason, the search backtracks beyond that point (removing $a(\mathcal{A}_{CGA}, v)$ from D_v), then another variable might be selected instead of v. Like this, *assignments* and not just variables are *dynamically ordered* throughout the search process in such a way that those decisions contained in $\mathcal{A}_{CGA}$ will always be taken as early as possible.

2. Discrepancy-based search methods are used to express the belief that the assignments in $\mathcal{A}_{CGA}$ are probably good ones. That is, we try to stick to the decisions made by CGA, and we would like to make only few deviations. In our implementation, this issue is handled by using a variant of the LDS search method. In the original LDS proposal, based on the assumption that heuristic decisions are less accurate high in the search tree, early decisions are reconsidered first. But in our case, the assignments for variables in V_2 are realized in the beginning, thus the contrary holds. Therefore, we prefer to use mEDS in this phase, too.

We should also note that the function HTSOPTIMIZE in line 1 of algorithm 1 might or might not use LNS. That also holds for line 8, where LNS is mentioned explicitly.

That choice should be tuned towards the specific case. LNS as a stand-alone method is not preferred due to the fact that it is likely to get stuck in a local optima. However, for our purposes, the most reasonable choice would be to use LNS after finding only one solution with a global tree search method. The local optimum will not be a problem, since the main optimization steps will follow, and much time will be gained. In any case, the user should use the method that provides a relatively good solution in the shortest time possible. We show the effect of such a choice in our experimental results. We also use it in line 8 to overcome a problem that might arise when bounding the values of the variables in V_3: Variables in V_3 belong to assignments which have the same values in both $\mathcal{A}_{HTS}$ and $\mathcal{A}_{CGA}$. And they are bound to their values as proposed by CGA to explore promising regions of the search space. However, this might not be true in all cases. Thus, using LNS on $V_1 \cup V_2 \cup V_3$ instead of only $V_1 \cup V_2$ might help on reviewing some almost certain decisions that might not be as accurate. Of course, it is still a matter of choice to use or not to use LNS and if so, to use it on $V_1 \cup V_2 \cup V_3$ or only on $V_1 \cup V_2$. In our experiments, we used LNS with mEDS as the subtree search method.

4.2. Second way of integration: generating combinable columns and exploiting dual values

We propose a second integration strategy, that is applied on company B. In this case, the convergence of the CGA approach towards an optimal solution is assisted by HTS first by constructing a set of initial columns that are combinable to complete partitioning solutions in a startup phase, and secondly by constructing columns with negative reduced costs during the main optimization phase. These columns are guaranteed to be extendible to a feasible solution, since they are extracted from one. A top level sketch of this method appears in algorithm 3. $\mathcal{C}$ is a set of rosters, $\mathcal{A}$ is an assignment, and *duals* are the dual values corresponding to this assignment (obtained by the CGA). HTSPOSTNRC posts a constraint on the number of rosters with negative reduced costs.

Algorithm 3. Top level algorithm for the second method

```
1. $\mathcal{C} \leftarrow$ HTSTREESEARCH($V$, DEFAULTSVAR, DIVERSESVAL)
2. **repeat**
3.    $\mathcal{A}$, *duals* $\leftarrow$ CGAOPTIMIZE($\mathcal{C}$)
4.    HTSPOSTNRC(*duals*)
5.    $\mathcal{C} \leftarrow$ HTSLNSTREESEARCH($V$, MAXDUALVAR, MAXDUALVAL, $\mathcal{A}$)
6. **until** stopping condition
```

4.2.1. Startup heuristic

In the CGA, columns are generated for each crew member sequentially. By using dual information, columns with negative reduced costs are generated. Thus, when the problem is non-degenerate, they lead to a decrease in the continuous relaxation of the master problem. Therefore, to find high quality rosters, "good" dual values are needed. Especially in the beginning, the information contained in the dual values is very poor. This is because usually no feasible solution is known at this point, and penalties stemming from dummy columns (that have to be introduced in the master problem to guarantee the existence of a solution) have a great impact on the dual values. We need to find a set of rosters that can legally be combined to form a set partitioning solution to the ACA. However, the column generator of the CGA is hardly able to produce such a solution, as it computes one roster at a time and is only indirectly aware of colliding pairings in different rosters.

Algorithm 4. Modified value selection heuristic for the second method

```
DIVERSESVAL($V$, $v$, $A$, $k$)
1. $val \leftarrow$ NIL
2. **repeat**
3.    $val \leftarrow$ DEFAULTSVAL($S$, $v$)
4.    **if** the assignment $\langle v, val \rangle$ appears more than $k$ times in $A$ **then**
5.       remove $val$ from $D_v$
6.    **else**
7.       **return** $val$
8. **until** $val \neq$ NIL or $D_v$ is empty
```

HTS can help here. In an integrated approach, it is used to generate a bunch of complete feasible solutions in the beginning, thereby providing one column for each crew member with every schedule found. Thus, a first set of columns that we know can be feasibly combined to a complete Set Partitioning solution provides the CGA with the necessary “grip” to accelerate towards promising parts of the search space with respect to the “real” objective without disturbing penalties.

Line 1 of the algorithm 3 realizes this idea. HTS searches for an initial number of solutions without performing optimization. The number of solutions to be found is a parameter that has to be tuned with respect to the time spent in this phase and the quality of the initial dual values.

Another parameter that has to be taken into concern is the diversity of the columns generated. It may be desirable to have many diverse rosters at hand that allow more and more profitable combinations in the master problem. One rule of thumb used in practice is that no crew-pairing assignment should appear more than a certain number of times in these columns. This restriction is taken into account by the slightly modified value selection heuristic DIVERSESVAL, which appears in algorithm 4. It works exactly as the value selection heuristic that would normally be used, but it also records the assignments made and limits the number of times a crew member can be assigned to a pairing.

In algorithm 4, A is the current set of solutions. Each time a solution is found by HTS, this solution is stored in A. Before assigning the value that would normally be selected by DEFAULTSVAL in line 3, it is checked whether the assignment appears less than k times in A. Otherwise, the value is removed from v's domain. This heuristic, in coordination with Depth-Bounded Discrepancy Search, see [18], guarantees that columns will be adequately different from each other to make the CGA method even more efficient.

Especially for large data sets, many initial solutions are needed. To speed up their computation, we try to shrink the search space: First, only one solution is computed. Then, the LNS search procedure is applied to obtain solutions that satisfy the diversity conditions only in local areas of the search space. For example, time windows can be used to limit the search space.

4.2.2. Main optimization loop

As shown in line 3 of algorithm 3, CGA performs an optimization run taking the columns produced by HTS as input. It returns an assignment $\mathcal{A}$ as well as the corresponding dual values for the crews and pairings. The solution returned is feasible with respect to all the company's rules and regulations. Then, starting from this point, HTS performs a locally limited search for columns with negative reduced costs.

The constraint posted in line 4 of the algorithm asserts that a certain number of the columns corresponding to each solution found will have negative reduced costs. This number is defined empirically. Finding a schedule that consists of columns with negative reduced costs only is rather unlikely. On the other hand, producing only few such columns is a wasted effort. Our experiments showed that schedules consisting of 30% columns with associated negative reduced costs can be achieved for our test set. But that does not mean that 70% of the columns produced are garbage! Instead, those columns guarantee that all newly generated columns can be extended to a feasible solution. Thus, the additional columns produced are important with respect to integer feasibility, whereas the columns with negative reduced costs reflect our search for improving solutions with respect to a linear objective.

Line 5 performs an LNS search with few deviations regarding the solution provided by CGA. The pairing with the maximum dual is assigned to the crew with the maximum dual as long as this crew member's reduced cost is not guaranteed to be negative already. Again our search method of choice is mEDS.

5. Numerical results

To demonstrate the superiority of combined approaches integrating CP and OR techniques, we applied the hybrid algorithms as presented above to real-world crew assignment problems (see section 2.2). Both the CGA and the HTS are prototype implementations only. Within a research project, it is not realistic to develop implementations for the ACA that could compete with the best industrial codes regarding overall speed, because those codes were produced during hundreds of person-years. Therefore, we just try to circumstantiate the gain in efficiency that can be obtained when combining methods from OR and CP.

We applied each method integrating HTS and CGA on the airline cases that motivated their development. All algorithms were implemented in C++ on top of Ilog software [9,10]. The first integration strategy was applied on two monthly data sets from company A. Experiments for this case were performed on a 640 MB, 296 MHz SUN

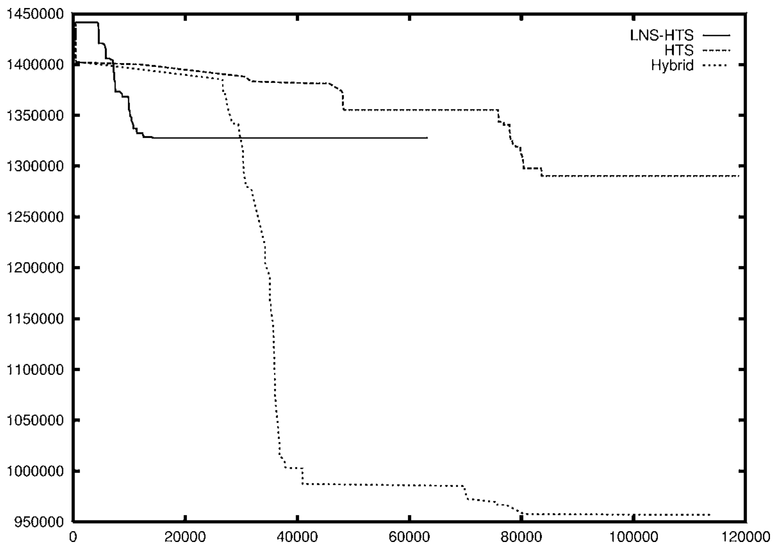

Figure 2. Data set with 65 crew members and 959 pairings.

UltraSPARC-II, with a time limit of 120 000 seconds.[3] The efficiency of our algorithm improves on the production system which company A currently uses.

Figure 2 is a cost (i.e., dissatisfaction) versus time graph showing the performance of the hybrid and the pure HTS methods applied on a monthly data set containing 959 pairings and 65 crew members. The problem is stated as a minimization problem. The curve marked "LNS-HTS" corresponds to a hasty strategy in which, after one solution is obtained, LNS is used to achieve some good solutions quickly. The "HTS" curve shows a more mature strategy, where the search finds several good solutions before LNS is applied to locally optimize them. The curve marked "hybrid" shows the performance of the hybrid approach, which clearly outperforms both. Interestingly, the pure CGA cannot detect any feasible solution at all. Within 120 000 seconds, it was not able to remove all dummy columns from the solution, i.e., the original master problem without dummy columns still is infeasible.

In these specific experiments, for exhibition purposes only, we call the HTS strategy in line 1 of algorithm 1 to show that it would have the best performance regardless of the startup phase. That is the reason why "LNS-HTS" outperforms "hybrid" in the beginning. Of course, we repeat that a reasonable choice for the startup phase of algorithm 1 would be a strategy more like "LNS-HTS". This strategy is used in the experiments of figure 3, which shows the performance of the same methods on another company A monthly data set containing 766 pairings and 50 crew members.

The following set of experiments is carried out to investigate the second way of integration. Experiments were performed on a 128 MB, 143 MHz SUN UltraSPARC,

[3] Curves stopping before this threshold indicate that no better solution was found from the moment corresponding to the end of the curve until the time limit has been reached.

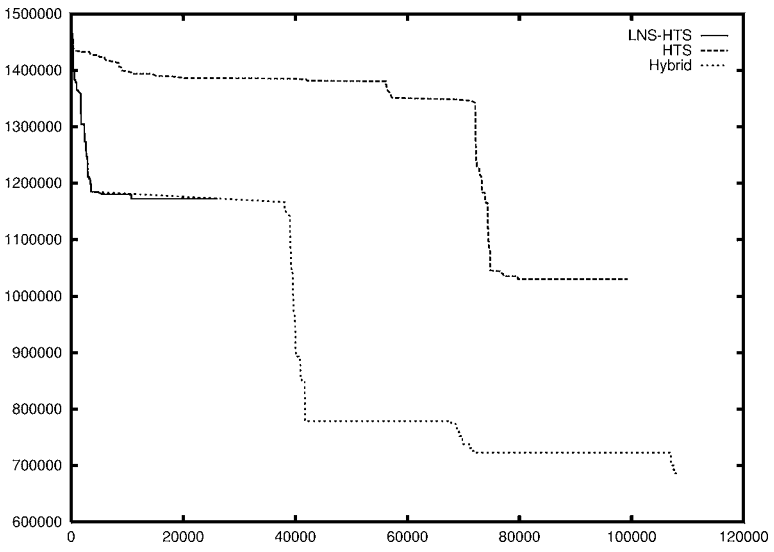

Figure 3. Data set with 50 crew members and 766 pairings.

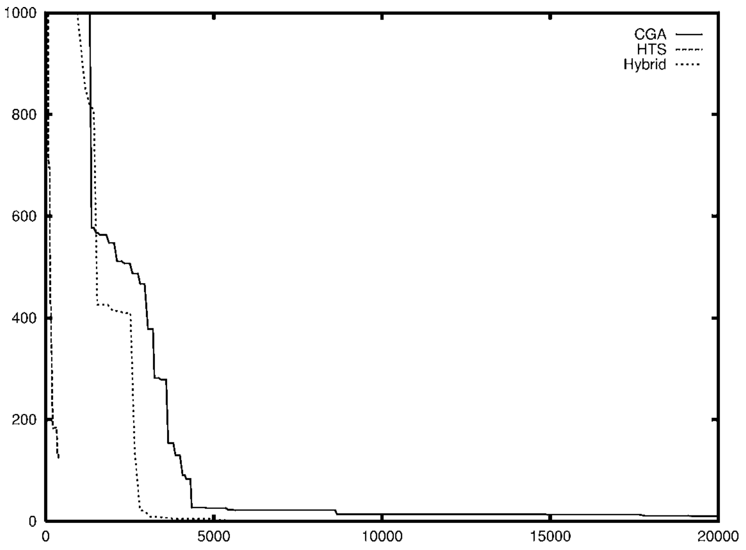

Figure 4. Data set with 7 crew members and 129 pairings.

with a time limit of 20 000 or 70 000 seconds depending on the problem size. Figure 4 shows the costs versus time plot for CGA, HTS and the second, so called, *consolidated approach* for a data set with 7 crew members and 129 pairings. Initially, HTS generates a solution and passes it over to CGA, which performs one optimization iteration. The resulting schedule is passed back to HTS, which rebuilds it and then locally searches for solutions containing as many rosters with negative reduced costs as possible. The PostNrc constraint guarantees that an adequate number of such rosters will be re-

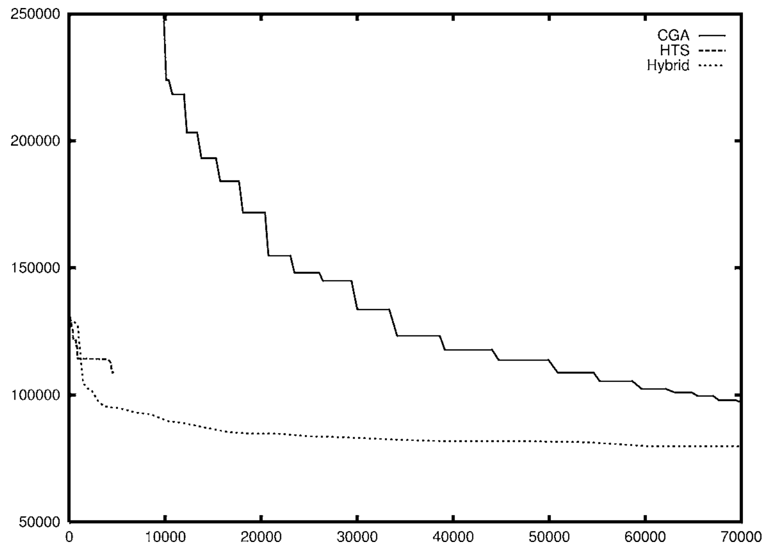

Figure 5. Data set with 30 crew members and 279 pairings.

turned. These rosters are then passed back to the CGA, and the process is repeated.

The same approach is used on a bigger problem instance, as shown in figure 5. The plots depict the expected behavior of CGA and HTS. CGA steadily optimizes the objective, but the quality of the initial solution is poor. Moreover, the time needed to find a first solution grows with the problem size. On the other hand, HTS finds relatively good solutions quickly by using heuristic information, but soon gets stuck. The consolidated approach benefits from both approaches: it finds good solutions quickly because of HTS and then steadily continues to refine the solutions due to the help of CGA.

It can also be seen that the integrated approach is slower than HTS early in the experiments. During that time, the hybrid approach is using the HTS module to create an initial set of columns according to the startup heuristic. The reason why HTS is slower in the consolidated case is that the goal is not to find better and better solutions, since the main optimization burden lies on the CGA side. Instead, HTS rather tries to find diverse rosters, which help CGA to find better solutions in the following.

The experiments regarding the second way of integration show that it is always useful to assign the task of finding a set of initial solutions to the HTS approach. The best number of solutions computed initially depends on the rule set as well as on the characteristics of the instance. Assigning the main optimization burden to CGA is the default choice, as it views the problem globally taking into account all variables and constraints at a time. If minor local adjustments can lead to quality improvements, then having HTS perform LNS searches throughout the process is cost effective. Furthermore, if the column generation process gets stuck, i.e., if a significant number of columns with negative reduced costs proves not be combinable to an IP solution, then having HTS generate solutions incorporating columns with negative reduced costs is cost effective, too.

5.1. Combining the methods

Numerical results clearly show that each hybrid approach is successful on the airline case on which it is applied in our experiments. The question that arises is whether the two hybrids can generally be combined or not.

We believe that orthogonality generally holds: A meta-hybrid could start off by having the HTS construct a set of solutions out of which diverse and feasibly combinable columns can be extracted. Then, the CGA approach can be used to improve on a relaxed version of the problem, which is repaired by the HTS approach.

We found that whether or not the use of one of the hybrid approaches we presented can speed up the computation of a good solution is problem dependent:

- Of course, the first hybrid can only be applied profitably, if the master problem is hard enough to justify the use of a relaxation that must be repaired at some point. Regarding airline case B, this precondition is not fulfilled, which is why we cannot apply hybrid 1 on this case.
- Using initial solutions provided by the HTS approach, in order to speed up the starting phase of CGA, only pays off when the CGA approach alone has difficulties in driving dummy columns out of the basis or spends too much time on this phase of the process. This is not given in airline case A, which causes that hybrid 2 cannot be used profitably here.

We conclude that generally the two hybrids can be combined, but the usefulness of a meta-hybrid is problem dependent. And its tuning heavily relies on inherent problem properties, which might not be known a priori.

6. Conclusions

For the ACA as an example, we have shown how CP and OR techniques can help each other to overcome their fundamental weak points. We believe that the ideas discussed in this paper can be generalized for other problems as well, especially in connection with (CP based) column generation. We presented results on large scale real world ACA data, which show clearly visible improvements in performance of the hybrid approaches compared to the solitary methods.

While OR methods view a problem globally and show a good ability to detect promising regions of the search space, CP methods can efficiently handle feasibility problems and are well suited to resolve local conflicts. The first way of integration tries to combine these advantages. It uses the CP based Column Generation approach (CGA) to compute cost efficient yet relaxed solutions to the problem, and then resolves conflicts of overcovered pairings by applying a heuristic CP tree search (HTS). The synergy effects are particularly visible if a lot of work has to be grouped in relatively few partitions. Then, column generation alone often fails to generate combinable rosters, and the use of HTS as a repairing module helps a lot to increase the overall performance.

The second way of integration that we introduced concerns the use of dual values. We showed how column generation approaches can profit from CP via the computation of diverse combinable initial columns. On the other hand, the use of dual information in a CP based heuristic tree search has shown to be very efficient. It allows to laden the optimization burden on the OR part and away from CP, which then can focus on what it was designed for originally, namely to solve constraint satisfaction problems.

References

[1] C. Barnhart, E.L. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh and P.H. Vance, Branch-and-price: Column generation for solving huge integer programs, Operations Research 46(3) (1998) 316–329.

[2] A. Caprara, F. Focacci, E. Lamma, P. Mello, M. Milano, P. Toth and D. Vigo, Integrating constraint logic programming and operations research techniques for the crew rostering problem, Software – Practice and Experience 28(1) (1998) 49–76.

[3] A. Caprara, P. Toth, D. Vigo and M. Fischetti, Modeling and solving the crew rostering problem, Operations Research 46(6) (1998) 820–830.

[4] P.R. Day and D.M. Ryan, Flight attendant rostering for short-haul airline operations, Operations Research 45(5) (1997) 649–661.

[5] T. Fahle, U. Junker, S.E. Karisch, N. Kohl, M. Sellmann and B. Vaaben, Constraint programming based column generation for crew assignment, Journal of Heuristics 8(1) (2002) 59–81.

[6] M. Gamache, F. Soumis, D. Villeneuve, J. Desrosiers and E. Gélinas, The preferential bidding system at Air Canada, Transportation Science 32(3) (1998) 246–255.

[7] M.R. Garey and D.S. Johnson, *Computers and Intractability* (Freeman, New York, 1979).

[8] W.D. Harvey and M.L. Ginsberg, Limited discrepancy search,. in: *Proceedings of the IJCAI '95* (1997) pp. 607–613.

[9] ILOG, ILOG SOLVER. Reference manual and user manual. V4.4, ILOG, 1999.

[10] ILOG, ILOG CPLEX. Reference manual and user manual. V6.5, ILOG, 1999.

[11] U. Junker, S.E. Karisch, N. Kohl, B. Vaaben, T. Fahle, and M. Sellmann, A framework for constraint programming based column generation, in: *Proceedings of the CP '99*, Lecture Notes in Computer Science, Vol. 1713 (Springer, Berlin, 1999) pp. 261–274.

[12] R.A. Rushmeier, K.L. Hoffman and M. Padberg, Recent advances in exact optimization of airline scheduling problems, Technical Report, George Mason University (1995).

[13] D.M. Ryan, The solution of massive generalized set partitioning problems in aircrew rostering, Journal of the Operational Research Society 43(5) (1992) 459–467.

[14] J. Schulze and T. Fahle, A parallel algorithm for the vehicle routing problem with time window constraints, Annals of Operations Research 86 (1999) 585–607.

[15] M. Sellmann, K. Zervoudakis, P. Stamatopoulos and T. Fahle, Integrating direct CP search and CP-based column generation for the airline crew assignment problem, in: *Proceedings of the CP-AI-OR '00*, Paderborn Center for Parallel Computing, Technical Report tr-001-2000 (2000) pp. 163–170.

[16] P. Shaw, Using constraint programming and local search methods to solve vehicle routing problems, in: *Proceedings of the CP '98*, Lecture Notes in Computer Science, Vol. 1520 (Springer, Berlin, 1998) pp. 417–431.

[17] P. Stamatopoulos, G. Boukeas, K. Zervoudakis, V. Stoumpos and C. Halatsis, Parallel CP-based direct crew rostering, PARROT Deliverable D-TEC2.1, University of Athens, University of Paderborn (1999).

[18] T. Walsh, Depth-bounded discrepancy search, in: *Proceedings of the IJCAI '97* (1997) pp. 1388–1393.

[19] G. Yu (ed.), *Operations Research in the Airline Industry*, International Series in Operations Research and Management Science, Vol. 9 (Kluwer Academic, Dordrecht, 1998).