Speeding up column generation by heuristically limiting the solution space for the airline crew rostering problem.

M. Zhang, A. Raith, A. Mason
Department of Engineering Science
University of Auckland
New Zealand
mzha612@aucklanduni.ac.nz

O. Weide
CAE, Auckland
New Zealand

Nov 2022

Abstract

The airline crew rostering problem involves scheduling tasks for each employee. Tasks may include flight pairings (sequence of flights that start and end at the same crew base), standby and reserve, with rest periods and days off in between. Difficulties in creating a suitable roster arise from deciding between all the possible task-employee combinations, which must also respect the airline industry's many rules, regulations, and self-imposed restrictions.

We can model the crew rostering problem as a set partitioning problem. Due to the large formulations that arise, we solve the crew scheduling problem with column generation, commonly solved as resource constrained shortest path problems (RCSPP). Even with column generation, it is a difficult problem to solve, as finding the optimal solution is intractable due to the exponential nature of the problem. So, we must apply acceleration techniques and heuristics to obtain a good solution in a reasonable time frame.

We introduce airline crew rostering, outline column generation and the difficulties in finding good solutions. We then present strategies to artificially restrict the number of tasks seen by each RCSPP to reduce the size of the problem and, with it, the computational burden on the overall solution process. We also highlight these effects when applied to a real-world crew rostering problem instance.

1 Introduction

Airlines are businesses that operate flights that transport people and freight from one destination to another. The airline industry is highly competitive, so reducing costs is important for an airline to maintain a market edge over its competitors. The highest cost for an airline is fuel (27%), and the next highest is the crew (23%) (Belobaba, 2009). Crew costs can be reduced by organising work for employees in an efficient manner, as well as retaining staff through high-quality work schedules.

Airline crew scheduling is the problem of assigning crew to work for all flights while respecting the airline industry's many rules, regulations, and self-imposed restrictions (rules). The highly combinatorial nature of the problem, coupled with all the rules, makes airline crew scheduling a complex problem to solve. Traditionally, airlines have solved it by hand (many small airlines still do). Airlines have increasingly adopted computers (and optimisation software), allowing the crew scheduling problem to be formulated and solved as an optimisation problem. Scheduling of airline crew is typically solved in two sequential problems, the crew pairing problem and the crew rostering problem.

A pairing is a sequence of flights that start and end at the same crew base (also known as a tour of duty). The airline crew pairing problem looks to create anonymous (not assigned to a specific crew member) pairings to cover all the flights the airline is flying while respecting all the rules. The majority of the cost saving comes from creating efficient pairings. The pairings created from the crew pairing problem form the work tasks that the crew rostering problem assigns to a specific crew. They are combined with other work tasks, such as standby and reserve, with non-work tasks, such as rest periods and days off, to form a complete rosterline for each airline employee. The crew pairing and the crew rostering problems result in a complete set of rosterlines for each crew member that covers every flight the airline operates for the rostering period (typically a month-long) while respecting all the rules.

One company that offers airlines a crew scheduling optimisation service is CAE. In this paper, we attempt to improve an aspect of CAE's crew rostering solver, the restricted networks heuristic. Section 2 presents the crew rostering problem and column generation as the solution method. Section 3 introduces the column generation subproblem and the restricted network heuristics. Section 4 steps through our computational experiments and insights, with Section 5 ending with us discussing our future direction.

2 Crew Rostering Problem

The crew rostering problem (CRP) can be formulated as a generalised set partitioning problem with upper-bound constraints as follows:

$$
\begin{aligned}
\text{[CRP]} \quad \text{minimise} \quad & \sum_{j \in R} c_j x_j & & \\
\text{subject to} \quad & \sum_{j \in R} a_{tj} x_j = b_t & \forall\, t \in T & \quad (1) \\
& \sum_{j \in R_e} x_j \leq 1 & \forall\, e \in E & \quad (2) \\
& x_j \in \{0, 1\} & \forall\, j \in R &
\end{aligned}
$$

Here E is the set of employees. T is the set of work tasks (pairings, standby and reserve) requiring employees to be assigned. R is the set of all feasible rosterlines (a feasible rosterline is one that satisfies all the rules) for all employees, where $R_e \subseteq R$ is the subset of rosterlines R specific to employee e. The binary decision variable x_j has value one if the jth rosterline is assigned to an employee. The objective is to minimise the total cost over all the employees' rosterlines. Constraints (1) are the generalised set partitioning constraints that ensure b_t employees are assigned to each task t. The variable a_{tj} has value one if task t is in the jth rosterline; otherwise, it is zero. Constraints (2) are the upper-bound constraints ensuring that each employee is assigned at most one rosterline.

The number of feasible rosterlines $|R|$ grows exponentially with problem size and quickly becomes computationally intractable. Rather than identifying the entire set of rosterlines R, we can consider a small subset of rosterlines $\tilde{R} \subseteq R$ and generate more as the solution process progresses. A restricted master problem (RMP) is a restricted version of the CRP formulation with R replaced by $\tilde{R}$.

The process of identifying suitable rosterlines (i.e., columns in the CRP formulation) to add to the set $\tilde{R}$ is the basis of column generation. Column generation is an iterative method that generates columns (rosterlines) while implicitly considering all the possible columns of the R. During column generation, the RMP is repeatedly solved with the current set of feasible rosterlines $\tilde{R}$. The optimal solution to the current linear programming relaxation of RMP provides information to guide the generation of columns in the next iteration in the form of dual values π_t of constraint (1) for each task t.

The column generation subproblem, which uses these dual values, is then solved to find feasible rosterlines with negative reduced cost to add into the set $\tilde{R}$ and then the RMP is resolved. This process is repeated until no negative reduced cost columns can be found, at which point the last solution found by the RMP is optimal for the original CRP.

The optimal solution obtained from the RMP is often fractional. Column generation is often embedded in a branch and bound framework (branch-and-price) to obtain an integer solution, a good review is given by Barnhart et al. (1998).

3 Employee Task Pools

Routing and scheduling problems (like the CRP) often give rise to column generation subproblems known as resource-constrained shortest path problems (RCSPP). In the context of CRP, the RCSPP looks to find the shortest path in a network comprising all the work and non-work tasks. The path is subject to constraints that represent the various rules. A feasible path from the source to the sink of the network will correspond to a feasible rosterline. A minimum cost feasible path represents a rosterline with the most negative reduced cost. All feasible paths found while finding a minimum cost path with negative reduced cost are then added to the feasible rosterline set $\tilde{R}$.

During the column generation process, we need to solve one subproblem for each employee, as each employee has a specific network with pre-assigned tasks and historical information related to feasibility with respect to the rules. We define an employee's task pool as the set of work tasks that the employee is allowed to generate rosterlines with. The RCSPP is an NP-Hard problem and thus becomes computationally intractable if we must consider a task pool containing all possible tasks.

We use heuristics to solve large-scale CRP problems and obtain any good quality solution in a realistic runtime. A restricted networks heuristic is one of the many strategies available (Desaulniers et al., 2002; Gamache et al., 1999). It artificially restricts the RCSPP network size by reducing the employees' task pool size, thereby reducing the individual subproblem solution times and thus reducing the overall solution runtime.

Recent approaches regarding the restricted networks heuristic includes the work of Quesnel et al. (2022). Their method of selecting tasks to be in task pools is by training a neural network with artificial historical data to predict probabilities that the task will be in the employees' rosterline in a optimal CRP solution. For each employee, a proportion of tasks with the highest probabilities are added to the employee's task pool. Nillius (2022)[1] attempts the approach of Quesnel et al. (2022) but with real-world training data instead. They train with the previous month's CRP formulation and solution to predict the current months employee task pool. When compared to their testing data, they show that their model's employees task pools are no different than creating the task pools randomly.

We propose an alternative approach to improve CAE's existing restricted networks heuristic. Our approach involves solving an integer program, the employee

[1]The author cites Quesnel et al. (2022) as an inspiration.

task pool assignment problem (ETP). We formulate the ETP as follows:

$$
\begin{aligned}
\text{[ETP]} \quad \text{maximise} \quad & \sum_{t \in T} \sum_{e \in E} c_{te} y_{te} & & \\
\text{subject to} \quad & p_t^{\min} \leq \sum_{e \in E | t \in T_e} y_{te} \leq p_t^{\max} & \forall\, t \in T & \quad (3) \\
& 1 \leq \sum_{t \in T_e | t_{\text{dep.day}} = d} y_{te} \leq v & \forall\, e \in E, \forall\, d \in D & \quad (4) \\
& y_{te} \in \{0, 1\} & \forall\, t \in T, \forall\, e \in E &
\end{aligned}
$$

Here E is the set of employees, T is the set of work tasks to be rostered, where $T_e \subseteq T$ is the subset of tasks that employee e can work, and D is the set of days in the current rostering period to be solved. The binary decision variable y_{te} determines if task t is assigned to employee e's task pool. Constraints (3) ensure that every task t is assigned to at least $p_t^{\min}$ and at most $p_t^{\max}$ employees. Constraints (4) ensure that each employee's task pool has at most v different work task options starting on (dep.day) every day d in the rostering period. The reward c_{te} is the reward gained when task t is assigned to employee e's task pool. The objective of the ETP is to maximise the total rewards.

The task pool sizes are defined by parameters $p_t^{\min}$, $p_t^{\max}$ and v, where the values should be adequately chosen based on the problem's size. The specific tasks that will be allocated to each employee's task pool by the ETP are determined by the rewards c_{te}. The following sections highlight some of the reward values we experimented with, computational results, and strategies to improve on the presented work in the future.

4 Computational Experiments

We perform our experiments on a mixed haul (short and long haul) airline's technical crew over two different length roster periods, with specific problem characteristics shown in Table 1. Our first experiment uses the Mixed-Tech-31D-01, which has a rostering period of 31 days, a size which is typically solved in practice. We then conduct preliminary experiments on the smaller instance Mixed-Tech-20D-01.

The solution quality reported (CRP objective) is a weighted sum of the slacks for constraints (1) and (2) and the quality of the individual employee's rosterline with respect to the rules. So, the smaller the solution quality metric, the better the crew roster is. Note that we only report the runtime to solve the linear programming relaxation of the CRP.

Table 1: Problem instance characteristics for our anonymous airline data, with employees and flight tasks rounded to the nearest 100.

Name	Rostering Days	Employees	Flight Tasks
Mixed-Tech-20D-01	20	800	3000
Mixed-Tech-31D-01	31	800	4600

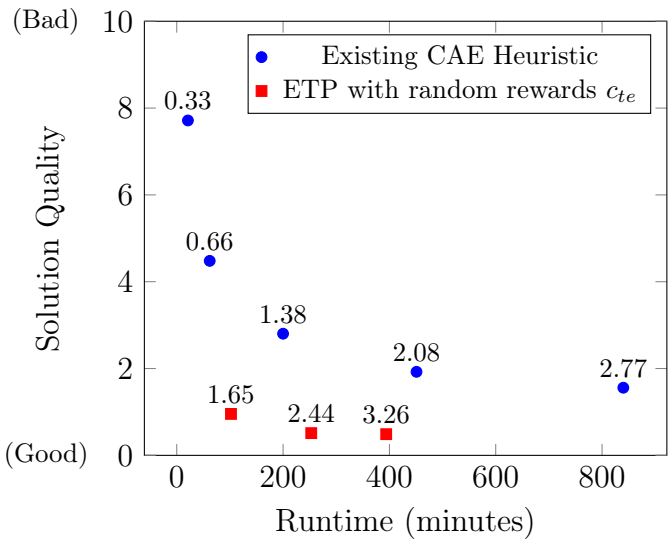

Figure 1: Mixed-Tech-31D-01 runtimes and solution qualities for different static task pools; the labels indicate the number of average work tasks per day $\bar{v}$ across all the employees' task pools.

4.1 Static Task Pools

We first compare the solution quality and runtime for problem Mixed-Tech-31D-01 with task pools found with CAE's existing heuristic and our ETP. For both approaches, we create the task pools before the column generation begins and we do not modify them during the solution process. We only change the size parameter for the CAE approach, whereas for the ETP, we change the maximum work task per day parameter v (with $v \in \{2, 3, 4\}$) and the randomly generated rewards values c_{te}. The rewards c_{te} for the ETP here are randomly generated as a uniformly distributed random value between 0 and 1 for every task-employee pair t, e.

Figure 1 illustrates how our task pools from the ETP perform against the existing CAE heuristic. We solve the Mixed-Tech-31D-01 problem with several different task pools sizes for each approach so we also see how the task pool size affects the solution quality and runtime. The labels in Figure 1 indicate the average number of work task per day ($\bar{v}$) value across all the employees' task pools, which will be lower than the ETP task per day value v due to employee preassigned tasks. The average task per day value $\bar{v}$ gives us a measure of the task pools' size after being created by the ETP (and CAE heuristic).

Within each approach, Figure 1 shows (as we have noted in Section 3) runtime is proportional to the size of the task pool. As the task pool size increases we also obtain better solution qualities (along side the increase in runtime) because the column generator is less restricted. Between tasks pools with similar average number of tasks per day $\bar{v}$, our results in Figure 1 show that our approach has a better solution quality and has faster runtime than CAE's heuristic. A possible reason for this is the fairly ad-hoc approach by CAE to create their task pools. These results prompt our research to explore whether we can improve our ETP, as we would expect task pools generated with carefully chosen reward values to outperform random reward values.

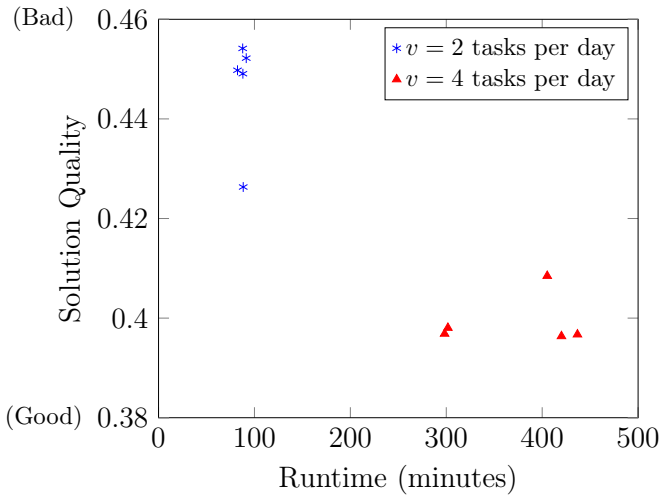

Figure 2: Mixed-Tech-20D-01 runtimes and solution qualities for multiple static task pools obtained from the ETP. Each task pool is created with different random reward values.

Figure 2 compares the runtime and solution quality for the problem Mixed-Tech-20D-01 between different task pools obtained from solving our ETP. It shows task pools with ETP parameters $v = 2$ and $v = 4$ with random rewards c_{te} uniformly distributed between 0 and 1, with five different sets of random costs generated for each v-value.

Within each v group we can see variation in solution quality and runtime, with more pronounced runtime variation in the $v = 4$ group. This indicates that specific task pools can obtain a better solution quality than others. The results support the idea that selecting certain combinations of tasks is better than others. It also highlights the need of making task pools as small as possible to keep runtimes low.

4.2 Dynamic Task Pools

Different rostering periods are often very different in practice. Factors such as seasonal variation in flights, employee preassigned flights or vacation and employees' history all contribute to making the current CRP very different from previous CRPs. So it can be difficult to identify good tasks for the task pools from previous data. We are attempting to avoid this hurdle by using information obtainable from only the current problem instance. Specifically, we will use information during the solution process to enable us to dynamically adjust our task pools (that we may have poorly created). The goal is to produce good CRP solutions while maintaining a short runtime by keeping the task pool small.

Figure 3 shows the results of two different strategies to add n tasks from the subset of tasks that the employee can work (T_e) to the task pool of each employee e. The tasks are added at the same column generation iteration (iteration 17) partway through the solution process for each experiment. One strategy is to add n tasks from T_e with the largest duals for constraint (1) π_t (from the current iteration), breaking ties randomly. The other strategy is to add n different tasks chosen randomly from T_e to each employee's task pool.

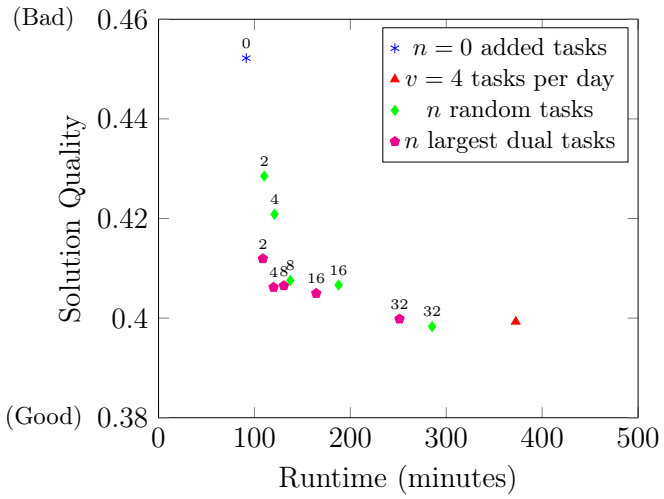

Figure 3: Mixed-Tech-20D-01 runtimes and solution qualities for the dynamic task pools obtained from the ETP. The labels indicate the number of additional tasks n added to the task pool during the solution process.

In Figure 3, as we increase the number of tasks added (n), the runtime increases, and solution quality improves as a consequence of the task pool size increasing, similar to Figure 2. Figure 3 also compares the task pool without added tasks (with $n = 0$) and the average solution quality and runtime for the $v = 4$ group from Figure 2. The results provide evidence that using information from the current solution can be beneficial towards identifying the best employee task pools. We chose the value $n = 32$ so the average tasks per day $\bar{v}$ after adding tasks to the task pool is comparable to the average $v = 4$ group from Figure 2 at $\bar{v} = 3.2$. The similar average tasks per day value (after tasks are added to the task pool) indicates that starting with a smaller task pool and adding tasks during the column generation leads to faster runtimes than starting with a bigger task pool size. Between the different strategies to add n tasks, aside from $n = 32$, the task pools modified with tasks based on dual values π_t obtain a better solution quality in a faster runtime than adding random tasks, indicating that information based on the current solution can be useful in improving an initial task pool.

We have also investigated changing the task pool dynamically after every column generation iteration. The first method uses the dual values π_t of constraint (1) for all employees e, as the reward c_{te} for the ETP. The intuition behind this is that the RMP indicates to the column generator that certain tasks are more 'needed' than others by their respective dual values for each constraint (1) and (2). By using the dual values as rewards, the tasks with higher dual values are more likely to be placed in an employee's task pool. Hence the column generator should be able to create more columns that include tasks with high dual values. The other method is randomly assigning the rewards c_{te}, where ETP is re-solved with different uniformly distributed random reward values in every column generation iteration.

Figure 4 shows that when we use dynamic task pools using the current duals π_t as rewards c_{te} for the ETP in every iteration, it performs both worse in solution quality and runtime than the static task pools from Figure 2. It also shows the dynamic task pools that are created with random rewards c_{te} perform better in

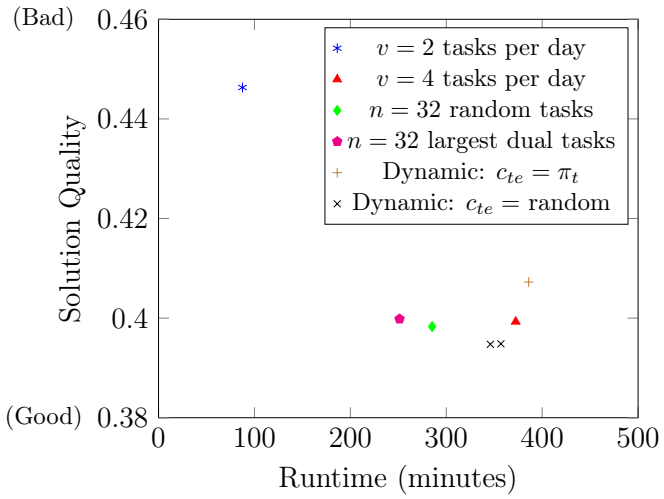

Figure 4: Mixed-Tech-20D-01 runtimes and solution qualities for the dynamic task pools by resolving the ETP with different rewards c_{te} every column generation iteration. It also shows the averaged results (across the 5 runs in each v-group) from Figure 2 and the $n = 32$ added tasks from Figure 3 for comparison.

solution quality and runtime than the static task pools from Figure 2. Between each other, using the duals π_t performs worse than using random rewards for the ETP. We currently do not have an explanation for this, but we suspect that completely changing the task pool every iteration may have an effect here.

5 Conclusion

We set out to improve CAE's restricted networks heuristic for the airline crew rostering problem. We formulated an ETP as an alternative strategy to create employee task pools. Through our initial experiments, we have shown that the ETP may create better task pools than CAE's current approach. However, more work needs to be done to identify the best strategy(s) with more extensive experiments to improve further the runtimes and solution qualities that can be achieved.

Our current results are limited to one particular problem instance (over different rostering periods), so we need to be certain whether our results are specific to a particular instance or can be generalised to all instances. So one of the next steps for us is to perform these same experiments over a series of different real-world instances to see if the trends we have seen remain the same. Furthermore, we can still explore many different strategies for extending the dynamic task pool approach, such as considering variations on when to adjust the task pool or which specific task pools to modify.

References

Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W. P., & Vance, P. H. (1998). Branch-and-Price: Column Generation for Solving Huge Integer Programs. *Operations Research*, *46*(3), 316–329. https://doi.org/10.1287/opre.46.3.316

Belobaba, P. (2009). Airline Operating Costs and Measures of Productivity. *The Global Airline Industry* (pp. 113–151). John Wiley & Sons, Ltd.

Desaulniers, G., Desrosiers, J., & Solomon, M. M. (2002). Accelerating Strategies in Column Generation Methods for Vehicle Routing and Crew Scheduling Problems. In C. C. Ribeiro & P. Hansen (Eds.), *Essays and Surveys in Metaheuristics* (pp. 309–324). Springer US.

Gamache, M., Soumis, F., Marquis, G., & Desrosiers, J. (1999). A Column Generation Approach for Large-Scale Aircrew Rostering Problems. *Operations Research*, *47*(2), 247–263. https://doi.org/10.1287/opre.47.2.247

Nillius, J. (2022). *Deep Learning in State of the Art Airline Crew Rostering Algorithms* (Master's thesis). Chalmers University of Technology and University of Gothenburg.

Quesnel, F., Wu, A., Desaulniers, G., & Soumis, F. (2022). Deep-learning-based partial pricing in a branch-and-price algorithm for personalized crew rostering. *Computers & Operations Research*, *138*, 105554. https://doi.org/10.1016/j.cor.2021.105554