

CTI's Approach to Rostering

TPAC Rostering

TPAC Rostering is CTI's optimising software component for solving the rosters problem. It allocates skill vacancies on activities to staff members, considering the cost of each allocation, and the cost and fairness of the overall roster, and supporting staff preferences and recency (which is a qualification to do a task depending on the time between now and the last time that task was performed) considerations.

Here we give details as to how this is done.

Component Architecture

TPAC Rostering uses a standard component structure and crewing XML format for both input and output. This format is easy to use and no customisation is required to represent a wide range of crewing problems within the aviation, rail and road transport sectors.

The graphical interface to the optimiser allows the end user to start, stop and monitor runs; to manage extract data and to perform what-if analysis. TPAC Rostering can be embedded within crew management systems or run stand alone within TPAC WorkBench. Within TPAC WorkBench the end-user has full visibility over the rosters produced by the component.

TPAC WorkBench

TPAC WorkBench is an environment for loading, preparing and solving optimisation problems in both "what-if" and production (real world) mode. Tools are provided for data extract, data validation and cleansing, solution analysis, statistics and KPIs. TPAC Rostering can operate within TPAC WorkBench enabling the user to explore the effect of different preferences, rules or other parameters.

TPAC Rules

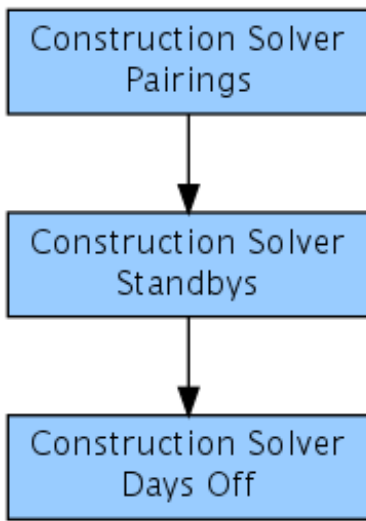
Business rules for TPAC Rostering are managed using the TPAC Rules system. This provides the efficient rule evaluation required by optimisation software, while allowing the rules to be viewed and modified by the end user in a readable form. The business rules themselves are not part of the component and can be changed at any time without requiring changes to the binary.

Configuration details are managed in a similar manner by the TPAC Rules system, including the large number of parameters which allow changes in solution procedure and in the specifics of how the algorithms work. This gives the software a high degree of flexibility and means that configuration can be changed on the fly as business requirements change or as the problem mutates from period to period.

Process Structure

TPAC Rostering consists of a number of components or "solvers", described hereafter, which can be connected in various different ways depending on the problem requirements; the construction solver, sequential solver, skill vacancy optimiser, fairshare optimiser and recency solver. A common, simple configuration is in Figure 1.

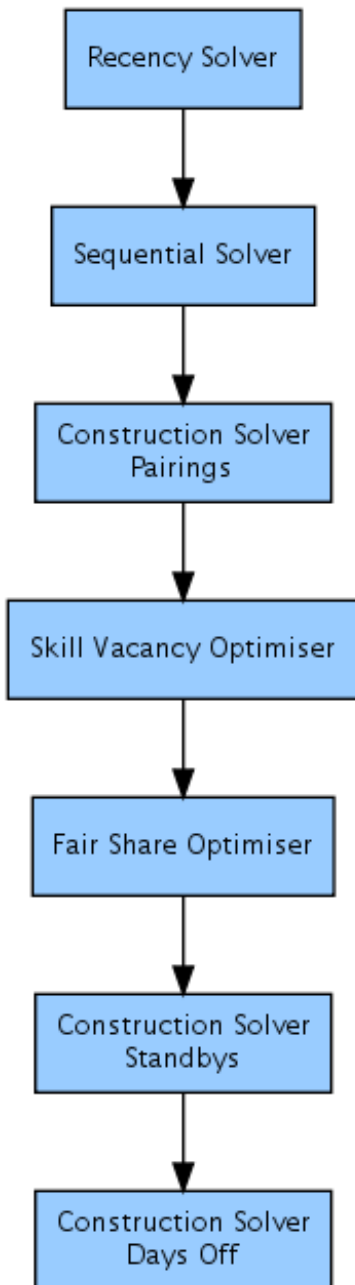
Figure 1. Simple Configuration



This configuration allows everything required by a simple operation. Fairshare is considered in the construction solver phase but for simple requirements it doesn't require a separate stage.

The most complicated configuration handles recency, bidding and fairshare; see Figure 2.

Figure 2. Complex Configuration



Construction Solver

The construction solver is designed to allocate large numbers of vacancies in an efficient manner. In a typical airline configuration, it may be used in three separate passes: to allocate pairing vacancies, standby vacancies and day off vacancies. This allocates the majority of vacancies in the problem, creating a feasible roster which will later be improved.

Algorithm

- Filter out unnecessary vacancies. Each pass of the construction solver is configured to consider a certain type of skill vacancy, filtering out all others; for example, to consider pairing vacancies but not those on standbys.
- Sort the list of vacancies. The sorting criteria are a configuration item designed by rostering experts at CTI in conjunction with the preferences of the customer. For example, it is usually desirable to allocate longer activities first, so these vacancies would be sorted first.
- Allocate the vacancies in the sorted list in order, using this procedure:

- Create a list of staff who are eligible to fill this vacancy. Eligibility considers whether they have the correct qualifications and whether their roster line has a gap which could fit the activity.
- Sort the list of staff. Once again, the sorting criteria are a configuration item; a common configuration is to sort those staff members with least work first, and another option might be to sort first those staff who could have recency renewed by this allocation.
- Try to allocate each staff member on the list, in order, to the vacancy.
- Use the configuration to determine when to stop the staff traversal. One common, fast option is to stop with and allocate the first staff member who can legally fill the vacancy. A slower but more accurate option is to try all the staff members and allocate the staff member who fills the vacancy cheapest.

Sequential Solver

The sequential solver is designed to allocate large numbers of vacancies while considering staff preferences, also called choices or bids. Choices can be either positive - that they would like a particular thing - or negative - that they would not like a particular thing. Choices can apply not just at the activity level but on any element of data that the system stores; for example, a staff member might prefer work which contains rest in Honolulu. When the sequential solver is used, it usually allocates the majority of vacancies in the problem, possibly in multiple passes, creating a feasible roster which is improved later.

Algorithm

- Filter out unnecessary vacancies. The sequential solver will usually consider only those vacancies for which bids can be made.
- Sort the choices. The sorting criteria are a configuration item; one common case is to sort more senior staff members' choices before those with less seniority.
- Split the sorted choices into groups of those which are no more important than each other; for example, two staff members with the same seniority may be treated as equally important.
- Satisfy the choices in this group using the following procedure:
 - For each choice in a group, find the set of activities which can satisfy it.
 - Sort the set of activities for each choice by the cost of allocating this activity to this choice.
 - Continually perform the cheapest allocation in the group, recalculating each time, until all choices in the group are filled or no more choices can be satisfied.
 - Consider other constraints in this process; such as that any individual staff member can only have a certain number of choices satisfied.
- The solver can also deallocate activities to match choices; in that case the procedure is identical to that quoted here but with allocation replaced by deallocation.

Skill Vacancy Optimiser

The skill vacancy optimiser is an improvement solver which uses swapping techniques to fill

unfilled vacancies. This is usually run immediately after the construction solver, to improve the roster by filling any gaps.

Algorithm

- Sort all remaining skill vacancies. The sorting criteria are a configuration item; one common case is to sort vacancies on more expensive activities first.
- Fill the vacancies in order using the following procedure:
 - Choose a subproblem containing the vacancy and a number of staff members who have the qualifications to be able to fill it.
 - For each chosen staff member, deallocate all the activities which overlap with the target vacancy.
 - Formulate a problem for the optimiser containing all of the chosen activities and staff members.
 - Solve the problem with the aim of always filling more vacancies than were originally filled; the optimiser may only fill the target vacancy but will also consider filling more than one vacancy if it is possible.
 - The optimiser will select, for the largest possible number of vacancies filled, either the cheapest solution or the most fair solution.

Fairshare Optimiser

The fairshare optimiser is an improvement solver which takes an existing roster and improves the overall fairness using swapping techniques, never attempting to fill extra vacancies. It can work with a hierarchy of fairness criteria. This is usually run after the construction or sequential solver and possibly the skill vacancy optimiser have been run, to improve the fairness of the solution.

Algorithm

- Sort the fairness criteria. The sorting criteria are a configuration item; for instance, block hours fairness might be more important than sharing of layover ports.
- Optimise each fairness criteria separately, though taking the others into account, using the following procedure:
 - Sort the staff members by the fairness according to this criteria.
 - Repeatedly choose the least fair staff member (often the furthest from the mean, for example) and improve their fairness using the following procedure:
 - Choose a subproblem containing a number of other staff members who have similar qualifications to this staff member such that on average the subproblem is fair.
 - Deallocate all activities for each staff member in the subproblem.
 - Formulate a problem for the optimiser containing all of the chosen staff members and their activities.
 - Solve the problem; the optimiser reallocates all the activities.

- The optimiser will select the solution with the best possible fairness with respect to the chosen criteria, and the cheapest cost.

Recency Solver

Most recency renewal tasks are handled during other stages, especially in the construction stage, where information is available as to what skills are renewed by an allocation, allowing it to be preferentially chosen. The recency solver is designed to handle the more complicated cases; for instance, training courses or chains of activities required. This usually runs before the construction solver on a small subset of activities.

- Filter out any activities which shouldn't be considered; for instance, pairings would usually be ignored in this special case phase.
- Gather the staff skills into skill sets, where those staff with identical skills are grouped together. Work out which skills in each set require renewing, when they should be renewed, and with what cost.
- Work out for each activity which skills it can renew.
- Match each skill which requires renewing with the activities which can legally renew it, and work out the cost for each of them.
- Either:
 - Input this set of renewals into the optimiser to get the optimal set of renewals, and then allocate these activities. This is not scalable and so is only feasible for small problems.
 - Sort the list of skills which can be renewed. The sorting criteria are a configuration item; usually the hardest to renew or most important will be sorted first. Allocate, in order, the cheapest renewal to each skill.

Further Information

You may wish to look at the [PDF version](#) of this document.