Tour Scheduling with Skill Based Costs
Abstract

Ed Mooney[1] and Tom Davidson[2]

[1] Montana State University, Bozeman MT 59717, USA
[2] Naval Undersea Warfare Center, Keyport WA 98345, USA

We propose an approach to the tour scheduling problem that neglects some elements of the general problem to quickly produce feasible tours that minimize hourly staffing costs. An efficient probabilistic local search algorithm solves a quadratic integer program representation of the problem. The resulting schedules may be used as templates to develop final schedules, or problem parameters can be refined to reflect days off and other requirements and the algorithm rerun.

We formulate the tour scheduling with skill based costs (TSSC) problem as a binary quadratic integer program. We solve TSSC over a planning horizon in days, $d = 1, \ldots, n_d$, with a fixed number of periods (e.g. hours) in a day. The set of time periods can then be indexed across the planning horizon by the set $T = \{t : t = 1, \ldots, n_t\}$. Furthermore, there is a known set of *functions*, or types of tasks, to be performed indexed by the set $F = \{f : f = 1, \ldots, n_f\}$.

The number of employees required for each function in each time period is specified in a n_txn_f requirements matrix with elements r_{ft}. The set of employee indexes, $E = \{e : e = 1, \ldots, n_e\}$, is also given. Employee e has skills qualifying him or her to perform a subset of functions, $F_e \subseteq F$ and is available for a subset of the time periods, $T_e \subseteq T$ specified by a *crossout matrix* obtained prior to scheduling. The subset of employees available to perform function f in period t, $E_{ft} \subseteq E$, can be be deduced from T_e and F_e.

We wish to minimize the total cost of assigning employees to satisfy the requirements and the number of non-adjacent assignments for employees within a day. The linear cost of assigning employee e to function f in period t is computed as the sum of the base hourly cost, c_1, plus the skill level increment, $c_2 \mid F_e \mid$, where $\mid F_e \mid$ is the number of functions employee e is qualified to perform. Hence, more skilled employees cost more per hour but allow more scheduling flexibility. Roughly equal daily shift lengths and contiguous periods within a day are encouraged by a quadratic term in the objective.

Hard constraints require that all staffing requirements be satisfied and employees assigned to at most one function per time period. *Soft* constraints represent the desire to restrict the assignments for each employee, e, to be within specified minimum and maximum total periods, $Pmin_e$ and $Pmax_e$, over the planning horizon.

The Dynamic Biased Sampling (DBS) local search algorithm was adapted to solve TSSC. DBS efficiently implements many of the advanced tabu search diversification and intensification mechanisms such as short and long term memory and aspiration criteria. Local search moves used to solve TSSC involve changing

the employee assigned to an f, t pair, or *slot*, to a feasible alternative. Candidate moves are sampled using a priority scheme that reflects the recency and frequency with which a slot's assignment has been changed. As applied to TSSC, the initial solution need not be feasible, and the moves need not be improving.

The DBS algorithm was applied to TSSC in a two phases. In the first phase, employee conflict-free assignments are found while improving the objective if possible. Once resource conflicts are removed, the second DBS phase attempts to reduce violations of the soft constraints and improve the objective function value while maintaining assignment and conflict feasibility. Lexicographic move evaluation is used where moves are ranked according to whether they (1) maintain hard constraint satisfaction, (2) maintain or reduce soft constraint violations, and, as a final tie braking criteria, (3) improve the objective function. A quadratic representation of the resource conflict constraint is used in the first phase of the heuristic and is dualized in the objective function. This formulation results in more efficient move evaluation. A problem generator was coded to create test problem instances with known characteristics and at least one feasible solution. The generator was coded in C and output problem instances for input to the search code. Input to problem generator included:

- Average demand (employees) per period, D
- Load factor, LF
- Average percent of periods employees unavailable (percent cross out), CO
- Percent deviation from the periods each employee is available, DEV
- Average percent of functions each employee is qualified to do, F

Output from the generator specifies the parameter values for an instance of the TSSC problem. The output includes staffing demand requirements for each function, the total number of employees and their attributes, as well as specified parameters such as the planning horizon length.

The search algorithm was run with 60 test problem cases and three instances of each case for a total of 180 instances. The load factor (LF) was set to 5, the deviation (DEV) was 1, and the number of functions was 5 for all instances. Similarly, all instances ran for seven 16 period (hour) days, or a total of 112 periods. The cases were defined by specifying the average levels of demand (D), employee cross-out (CO) fraction, and employee qualified function fraction (F). Average per-period demand was set at 5, 10 and 15. The average cross-out fraction (CO) levels were .2, .35, .5, for all demand levels. A cross-out fraction of .7 was also run for D=5. And, the average fraction of functions (F) used were .1 through .6 in increments of .1.

The cost parameters, c_1 and c_2 were set to 5 and .5 for all runs, and minimum and maximum employee hours constraints were relaxed. The runs were terminated after 5,000 iterations. Runs were made on a Dell 8200 with a 2.3 Ghz Pentium 4 processor running Windows XP professional. The code was compiled with Microsoft Visual Studio 6.0 in release mode and run under MinGW.

All (random) initial solutions had a large number of resource conflicts, which were removed in phase 1 very quickly. The average objective function improvement ranged from approximately 21% to 43%. The average time to the best

solution ranged from 2.12 seconds for one of the smallest cases to 79.85 seconds for one of the largest ones. The best solution was found close to iteration 5,000 in all cases, indicating that the search was still finding better solutions almost to the end.

Solution quality for all test cases was quite good with average gaps with a naive bound ranging from 3.4% to 14.4%. The bound was obtained with all constraints relaxed and the minimum cost choice made for each assignment. This bound is probably not very good, but was better for problems with less variability in the requirements and employee characteristics. The gap increased with problem size as indicated by demand and somewhat more dramatically with the average fraction of the functions an employee can perform. This is likely mostly due to a degradation in the bound due to the way the number of functions are sampled for employees.