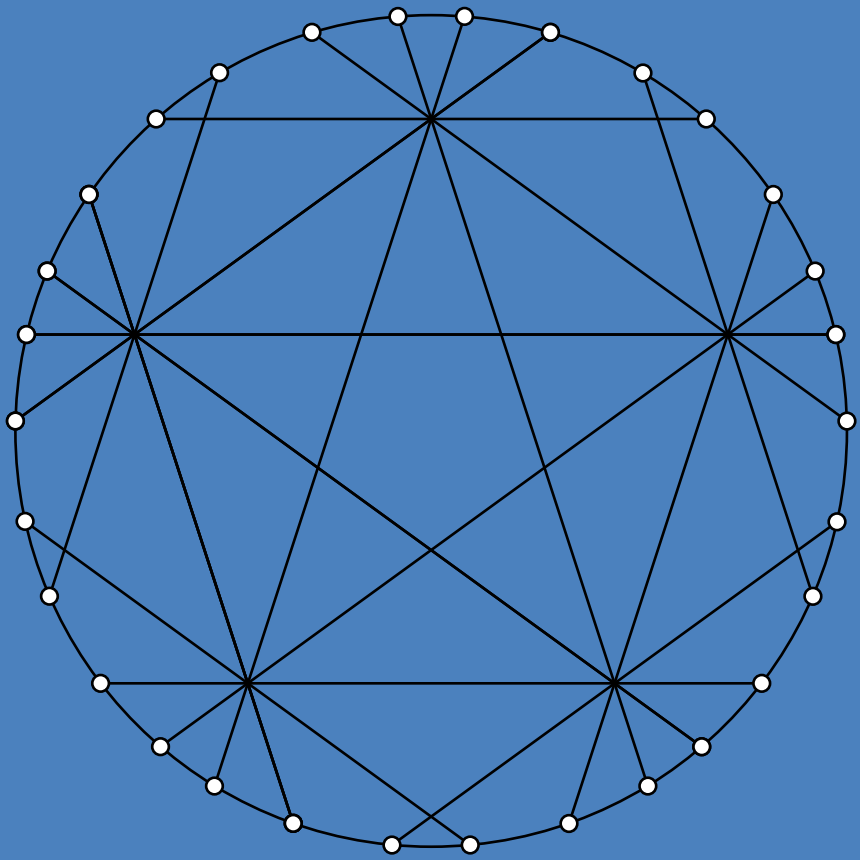
BULLETIN of the INSTITUTE of COMBINATORICS and its APPLICATIONS

Volume 80
May 2017

Editors-in-Chief: Marco Buratti, Don Kreher, Tran van Trung



Boca Raton, Florida

ISSN: 1183-1278

BULLETIN OF THE ICA
Volume **80** (2017), Pages 119–127

Classroom Note. Contributions of Graphs to Staff Rostering

Joy Lind

Department of Mathematics, University of Sioux Falls
joy.lind@usiouxfalls.edu

Abstract

Various models have been developed for the "rostering problem," that is, the problem of generating a feasible high-quality schedule ('roster') for staff members at a company or business. Such models are typically solved by integer-programming techniques. In this paper, we show how graphs contribute to an overall solution method for these types of problems. Our objective is to give students in an undergraduate discrete mathematics or graph theory course preliminary exposure to rostering problems and solution methods as well as to provide exercises that reinforce the graph application component.

1 Introduction

The "rostering problem" is the problem of generating a feasible high-quality schedule, or "roster," for staff members at a company or business. The roster includes, for each staff member, the sequence of shifts and days off to be worked by that staff member during the time horizon that the roster specifies. We use the terminology of [3] to describe the components of a roster. A *shift* is a period of time during which an employee works and is identified by a date, a start time, and an end time. For example, an employee may work a shift on Wednesday, February 15, that starts at 8am and ends at 5pm. For each staff member, we will seek to identify a *roster-line*, a sequence of shifts (days on) and days off during the roster period. Typically, rules will exist that define some roster-lines as *infeasible*, meaning that they cannot be assigned; an example would be a roster-line that

Received: 12 February 2017
Accepted: 14 March 2017

assigns too many days worked in succession without a day off. Otherwise, a roster-line is *feasible.* Each feasible roster-line has an associated *quality* measure that takes into account any number of quality objectives that staff members seek; for example, employees typically dislike split weekends in which one day in the weekend is worked, and the other is not. A *roster* is a collection of roster-lines, one for each staff member. A roster is *feasible* if it is comprised of feasible roster-lines and if it satisfies pre-specified staffing requirements, for example, that certain shifts be staffed with a minimum number of employees or that certain shifts be staffed with employees who have particular skills. The objective is to find a roster that is both feasible and high-quality. Meeting this objective can indeed be challenging; "the roster-line worked by each staff member is typically strictly governed by laws, union regulations and internal agreements... that can make it hard to create feasible rosters, let alone high-quality ones" [3].

2 A Mathematical Model for the Rostering Problem

The rostering problem is commonly modeled as an integer program and solved using a branch-and-bound technique. In this section, we provide a high-level overview of one such model that is developed in [3] then refer the interested reader to that supplemental reference for details. In this model, there are two sets of constraints: one set of constraints that forces the assignment of exactly one roster-line to each employee, and one set of constraints that ensures that the staffing requirements are met for each shift, while possibly permitting under-coverage or over-coverage on certain shifts. The model includes three sets of decision variables. The first set are binary decision variables λ_e^r defined for each employee e and roster-line r in the following way: $\lambda_e^r = 1$ if roster-line r is assigned to employee e and $\lambda_e^r = 0$ otherwise. The two remaining variable sets give the amount of under-coverage (slack) for each shift and the amount of over-coverage (surplus) for each shift. Each variable has an associated cost. The cost corresponding to the variable λ_e^r reflects the quality objectives and in particular, quantifies the extent to which employee e dislikes roster-line r. There is also a cost associated with each slack and surplus variable that reflects how important it is that the staffing requirements for a particular shift be exactly met. Typically, the slack and surplus variables are each bounded by values giving the minimum and maximum under-and over-coverage permitted. The objective is to minimize the sum of the costs of all assigned roster-lines plus the penalties from under-and over-coverage.

Even from the above high-level description, the reader may have already identified one significant challenge in formulating this model, namely constructing the complete set of feasible roster-lines for each employee. Fortunately, it turns out that this is not necessary. The solution method for the integer program described in [3] involves relaxing the integer constraints on the λ_e^r variables to create a linear programming (LP) problem, then applies branch-and-price, a branch-and-bound method in which at each node of the tree, a subproblem is solved to generate new columns to be added to the LP relaxation. We forego details about branch-and-price and column generation here and instead refer the reader to [1] and [2]. An important point, however, is that in this rostering model, generating columns is equivalent to finding feasible roster-lines. Thus, we need not construct a complete set of feasible roster-lines for each employee at the onset but rather develop (a subset of) them in the course of the branch-and-price framework. In the next section, we focus on the method for generating feasible roster-lines developed in [3] and show how graphs play a prominent role.

3 Generating Roster-Lines for Employees

Here, we summarize the method developed in [3] for generating roster-lines, with a focus on the graphical component. It will be helpful to define three additional terms related to shifts and roster-lines. An *on-stretch* in an employee's roster-line is a sequence of shifts that the employee works in succession. An *off-stretch* is a sequence of days that the employee has off, after an on-stretch. A *work-stretch* is an on-stretch followed by an off-stretch. One can think of a roster-line for an employee as a sequence of work-stretches. In [3], roster-lines are generated for each staff member according to the following three-step process: (1) on-stretches are generated from shifts, (2) work-stretches are generated from on- and off-stretches, and (3) roster-lines are generated from work-stretches. We next explain each of these three steps, then in Section 4, we provide a set of exercises that lead students through a clarifying example.

In step 1, we use graphs to generate on-stretches from shifts. One graph is constructed for each employee. In a graph, each shift is represented by a vertex, and a directed edge exists between two vertices when the corresponding shifts are allowed to be consecutive in an on-stretch, according to pre-specified rules (e.g. that the shifts be neither too close together nor too far apart). Since different employees will have different skills, it may be necessary to exclude certain vertices from an employee's graph corre-

sponding to shifts to which the employee cannot be assigned. Each edge is given a weight that is calculated within the branch-and-price process and that takes into account costs of shifts and possibly costs of shift transitions. Within each graph, the all-to-all shortest path problem is solved by solving a one-to-all shortest path problem from each vertex in the graph. For each one-to-all shortest path problem, the starting vertex is selected and then the graph is reduced to only allow on-stretches up to the maximum on-stretch length (5 shifts, for example); the maximum length is generally dictated by a law or other regulation. The one-to-all shortest path problems are solved (e.g. by Dijkstra's Algorithm); their solutions then translate into minimum-weight on-stretches for the particular employee. Feasibility checks can be applied to each on-stretch. Additionally, pre-specified dominance rules can be checked to determine if any on-stretches are dominated by (inferior to) others. On-stretches that are either infeasible or dominated are removed from consideration; all other of the minimum-weight on-stretches are sent to step 2.

In step 2, work-stretches are generated from on-stretches for each employee. Recall that a work-stretch is an on-stretch followed by an off-stretch. On-stretches are those found in step 1. Each off-stretch is described by a start time and duration. A work-stretch is formed by taking an on-stretch and appending an off-stretch. Again, feasibility and dominance tests are applied, which may result in work-stretches being eliminated from consideration. All remaining work-stretches are sent to step 3.

In step 3, we use graphs to generate employee roster-lines from work-stretches. In each graph (one per employee), there is a vertex for each day, and there is a directed edge for each work-stretch that extends between the vertices corresponding to the work-stretch's starting and ending dates. (Although to clarify, edges are typically drawn so that their terminal vertex is the day *after* the work-stretch ended [3].) Additionally, there is a source vertex and sink vertex that correspond to a pre-determined start day and end day of the roster. Work-stretch weights are calculated within the branch-and-price process and are assigned to the edges. We now solve a one-to-one shortest path problem from the source to the sink in the graph. This finds a minimum-weight roster-line (sequence of work-stretches) for the employee. We briefly note that the method in [3] applies techniques to remove work-stretches (edges) at the onset, to reduce the size and complexity of the graph.

4 Student Exercises: An Example

In this section, we lead students through an example of the above process for generating a roster-line for an employee. Assume that we seek a one-week staffing roster for 1am, Monday, June 1, through 1am, Monday, June 8, for a group of employees at a particular company. Consider a "day" as a 24-hour block comprised of the following three work periods: 1am-9am, 9am-5pm, and 5pm-1am. Thus, the roster consists of 21 distinct shifts over 7 days:

- Day 1 = Monday, June 1, 1am-9am; Monday, June 1, 9am-5pm; Monday, June 1, 5pm through Tuesday, June 2, 1am
- Day 2 = Tuesday, June 2, 1am-9am; Tuesday, June 2, 9am-5pm; Tuesday, June 2, 5pm through Wednesday, June 3, 1am

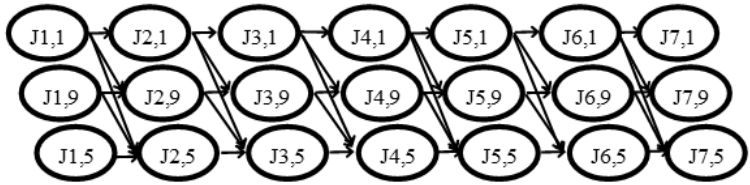
  ...

- Day 7 = Sunday, June 7, 1am-9am; Sunday, June 7, 9am-5pm; Sunday, June 7, 5pm through Monday, June 8, 1am.

Each shift has its own staffing requirements (that is, the number and type of staff members required for each shift). No employee can work more than 5 shifts without a day off. Also, shifts assigned to an employee cannot be too close together, according to the following rules: an employee can work at most one 8-hour shift per day, and if the shifts that an employee works contain a mix of shift types (for example, working 1am-9am on some days and 9am-5pm on others), the shifts must be "forward rotating" [4], which means that each successive shift starts at the same or later (but not earlier) time.

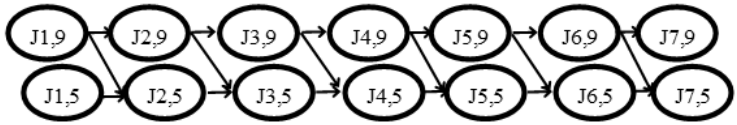
Exercise 1: Draw the graph representation of the on-stretch generation for an employee at this company.

Solution: We use the notation (JX,Y) to represent the shift that starts on June X at time Y. For example, (J1,5) corresponds to the shift that starts on June 1 at 5pm (and that ends at 1am on June 2). Note that edges exist only between consecutive days (and only between shifts that adhere to the "forward-rotating" rule); note that an edge that skips a day would be part of an off-stretch, since each skipped day would constitute a day off, and an edge between shifts on the same day would violate the rule that at most one 8-hour shift can be worked per day.



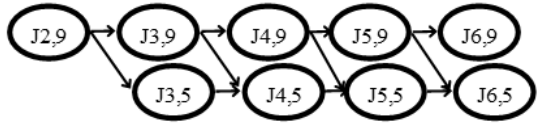
Exercise 2: Assume that Employee A does not have the skillset to be able to do the work required during the 1am-9am time period each day. Draw the graph representation of the on-stretch generation for Employee A.

Solution:



Exercise 3: Draw the subgraph of the graph in Exercise 1 for solving the one-to-all shortest path problem starting at (J2,9).

Solution: We reduce the graph from Exercise 1 to only allow on-stretches that start at vertex (J2,9) and that have at most 5 shifts.



Exercise 4: Assume that feasibility and dominance rules are specified and that weights are applied to the graph in Exercise 1. Assume that the

shortest-path algorithm finds the following minimum-weight on-stretches, which are verified to be feasible and non-dominated:

- (J1,9) → (J2,9) → (J3,9) → (J4,9) → (J5,9)
- (J3,1) → (J4,1) → (J5,1) → (J6,1) → (J7,1)
- (J2,1) → (J3,1) → (J4,9) → (J5,9) → (J6,9)
- (J1,5) → (J2,5) → (J3,5)
- (J4,9) → (J5,5)

Assume that off-stretches can consist of 1, 2, or 3 days. List the possible work-stretches for the employee in Exercise 1.

Solution: There are 5x3=15 work-stretches, where a work-stretch consists of one of five possible on-stretches (as given above), followed by one of three possible off-stretches (of length 1, 2, or 3 days).

- (J1,9) → (J2,9) → (J3,9) → (J4,9) → (J5,9) → 1 day off (day 6)
- (J1,9) → (J2,9) → (J3,9) → (J4,9) → (J5,9) → 2 days off (days 6 and 7)
- (J1,9) → (J2,9) → (J3,9) → (J4,9) → (J5,9) → 3 days off (days 6, 7, and 8)
- (J3,1) → (J4,1) → (J5,1) → (J6,1) → (J7,1) → 1 day off (day 8)

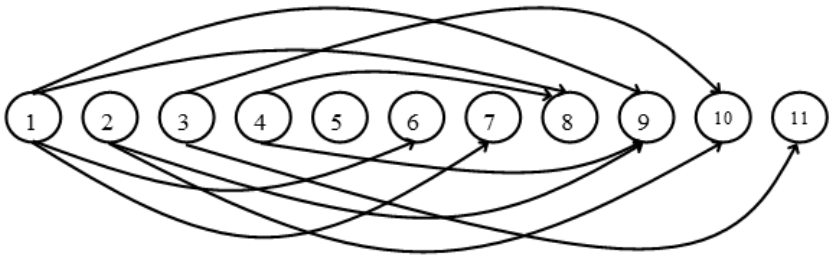
  ...

- (J4,9) → (J5,5) → 3 days off (days 6, 7, and 8)

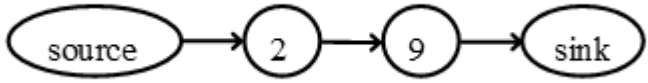
Exercise 5: Assume that among the above work-stretches, all of those with 1-day off-stretches are deemed infeasible or are dominated. Draw the graph representation of the roster-line generation problem for the employee in Exercise 1.

Solution: From the original 15 work-stretches, 10 remain after eliminating those with 1-day off-stretches. The resulting graph includes one vertex for each of the original 7 days with additional vertices for extra days needed to cover all of the work-stretches. There is one directed edge for each

of the 10 work-stretches; each edge starts at the vertex corresponding to the day when the associated work-stretch begins, and ends at the vertex corresponding to the day *after* the work-stretch ends. In the below graph, to assist with readibility, we exclude the source and sink vertices that are explained in Section 3.



Exercise 6: Assume that after weights are applied to the graph in Exercise 5, the below minimum-weight path is found from the source to sink. What is the corresponding roster-line for the employee?



Solution: This directed path in the graph corresponds to work-stretch: (J2,1) → (J3,1) → (J4,9) → (J5,9) → (J6,9) → 2 days off. This translates to the employee working from 1-9am on June 2, 1-9am on June 3, 9am-5pm on June 4, 9am-5pm on June 6, then taking June 7-8 off. Note that since this example used a very small time horizon for the roster period, it is unsurprising that the employee's roster-line consisted of a single work-stretch. Generally, rosters will be comprised of several weeks or months so that each employee's roster-line will contain multiple work-stretches.

References

[1] J. Desrosiers, M.E. Lübbecke, Branch-price-and-cut algorithms, in: Wiley Encyclopedia of Operations Research and Management Science, Wiley, 2010.

[2] J. Desrosiers, M.E. Lübbecke, A primer in column generation, in: G. Desaulniers, J. Desrosiers, M. Solomon (Eds.), Column Generation, Springer, New York, 2005, pp. 1-32 (Chapter 1).

[3] A. Dohn, A. Mason, Branch-and-price for staff rostering: An efficient implementation using generic programming and nested column generation, European Journal of Operational Research, 230 (1), 2013, pp. 157-169.

[4] A. Mason, D. Panton, Cyclic rostering using branch and cut, in: Proceedings of the 37th Annual ORSNZ Conference, University of Auckland, 29-30 November 2002, pp. 287-296.