Proceedings of the 1998 Winter Simulation Conference
D.J. Medeiros, E.F. Watson, J.S. Carson and M.S. Manivannan, eds.

ROSTERING BY ITERATING INTEGER PROGRAMMING AND SIMULATION

Shane G. Henderson
Andrew J. Mason

Department of Engineering Science
University of Auckland
Private Bag 92019
Auckland, NEW ZEALAND

ABSTRACT

We present a new technique (RIIPS) for solving rostering problems in the presence of service uncertainty. RIIPS stands for "**R**ostering by **I**terating **I**nteger **P**rogramming and **S**imulation". RIIPS allows great complexity of the stochastic system being rostered. This modelling freedom comes at a price, as the approach can be extremely computationally intensive. Therefore any reduction in computational effort using, for example, efficiency improvement techniques, is of great interest. We specify several ways in which these may be applied.

1 INTRODUCTION

In rostering applications, one wishes to minimise the costs of staffing, subject to the constraint of maintaining reasonable customer service. The prototypical example is the call centre. In these systems, customers dial a number, possibly join a queue of waiting customers, and then receive service from a service agent. The quality of a customer's service is usually defined in terms of a measure called "customer grade of service" (CGOS), which typically depends on the customer's waiting time in a queue. Some form of averaging may then take place to obtain an overall grade of service (GOS) for a given set of customers. The staffing problem is then to minimise staffing costs while ensuring some minimum average GOS is achieved during the rostering period. Other quality measures may drive the rostering problem. For example, when processing passengers at an airport (Mason, Ryan and Panton 1998), management typically require that some percentage of the passengers on each aircraft be processed within some stated time.

For the purposes of explanation, we shall assume that the staffing period is a day, although the typical staffing period is more likely to be a week or a month. This assumption is merely one of convenience and scale, and makes no substantial difference to the RIIPS approach.

Since customer loads can vary considerably over the day, and staffing decisions can typically only be made at discrete times, the day is broken up into several periods. Rostering decisions may only be made at the start of these periods.

Rostering problems with uncertain service requirements are typically solved in two phases (Mehrotra 1997). In the first phase, staffing requirements for each period of the day are determined using queueing models and/or simulation. If one assumes that the system reaches steady-state quickly, then within each period, steady-state queueing models provide excellent approximations to the number of service agents required. Unfortunately, fast convergence to steady-state is not typical for these systems. A second approach is to attempt to numerically calculate, or approximate, the time varying distribution of GOS. Strongly related ideas are discussed in, for example, Jennings et al. (1997). This approach has, at least to date, only been applicable to a restricted class of models. A third approach is to use simulation to obtain a required number of agents in each period of the day. This approach is taken in, for example, Eitzen (1994).

In the second phase, one attempts to build staff rosters that "cover" these staffing requirements using integer programming formulations of set covering problems. This two phase procedure, while an improvement over heuristic rostering, is not entirely satisfactory.

The two phase approach specifies a minimum number s_i say, of staff required to be on duty during time interval i in order to meet customer performance requirements. It is possible that by slightly modifying the staffing requirements, say by increasing s_i by one, and decreasing s_{i+1} by one, that the same degree of customer satisfaction is achieved. The modified solution may be "easier" to roster, in the sense that less staff are needed to fulfill the staffing requirements, or that the resulting roster is of a higher quality. However, because we specify a single set of staffing requirements in the first stage, such flexibility in rostering solutions is not achievable in the two phase approach.

The primary difficulty with this two phase approach is that it doesn't capture the *linkage* between adjacent time periods. A recent two-phase approach that attempts to capture some of this linkage is Thomson (1998). The approach is to use steady-state models to determine the performance of the system. The steady-state results are modified using heuristics to obtain approximations for time-dependent quantities. The staffing levels are obtained by solving a dynamic program with a heuristic cost structure designed to obtain staffing levels that are "easy" to roster to. The approach captures at least *some* of the linkage between time periods, but requires a tractable stochastic model of call centre operation so that the steady-state distributions can be computed. The RIIPS approach does not suffer from such a restriction, since simulation is used to obtain customer performance measures. It works as follows.

Suppose that the rostering period is broken up into p time intervals (periods) $1, 2, \ldots, p$. The start of these periods correspond to times when staffing decisions may be made, such as bringing extra servers on, or removing servers.

We begin by solving an integer program (IP) that provides the initial roster. Server allocations $\{r_i : i = 1, \ldots, p\}$ may be easily derived from the roster. Here, r_i is the number of available servers in period i.

Next, the r_i's are passed to a simulation, the day's operation is simulated (perhaps many times), and appropriate statistics are obtained. These results are translated into a GOS (or some equivalent measure) for each period. One or more "cuts" (additional constraints) are developed from the simulation output, and passed back to the rostering IP. The development of the cut(s) requires the knowledge of derivative information. This might be obtained using gradient estimation techniques such as likelihood ratio methods (Glynn and L'Ecuyer 1995) or Infinitesimal Perturbation Analysis (Fu and Hu 1997). Alternatively, one might use finite differencing to generate cuts. Finite differencing requires greater simulation effort than the above methods, but is more robust in that it applies to a larger class of models.

The rostering IP is then re-solved, and the process iterates (simulate—solve the IP—simulate—solve the IP $\cdots$) until some convergence criteria are satisfied. Typically the algorithm will conclude when the difference between upper and lower bounds on the objective is less than some tolerance. We do not enter into a discussion on how these bounds might be generated in this paper.

The RIIPS procedure allows great complexity in the simulation, as will be seen in Section 3, where we discuss how the cuts are generated. In particular, the primary assumption of the algorithm is that GOS in period i is a concave function of the available servers $r_1, \ldots, r_i$ up until period i. To see why this is reasonable, first note that the GOS should be an increasing function in each of the r_i's because, presumably, the addition of staff can only improve customer service. Next note that one would expect diminishing marginal returns as additional staff are added, so that the GOS should be a concave function of each individual r_j (holding other staffing levels fixed). Although this is not sufficient for the concavity we assume, it is certainly strong motivation for the assumption. In the case where GOS is not a concave function of staff allocations, the cuts generated by RIIPS are no longer necessarily valid. In this case the approach might be viewed as a heuristic that should be used with a degree of caution.

We remark that the essence of our approach is the proposal of a roster by an IP, the evaluation of the roster by some means, and the identification of an appropriate cut to pass back to the IP. It is not necessary that the evaluation be performed by the simulation. For instance, one could envisage the numerical solution of a partial differential equation at this step! In terms of defining a cut, the cuts proposed in this paper are not the only ones that might be considered. Other cuts might be identified, depending on the application at hand. In particular, if GOS is not necessarily a concave function of staff allocations, it may be possible to identify alternative cuts that must be satisfied by optimal solutions.

The remainder of this paper is organised as follows. In Section 2 we describe the problem in more detail in the context of call centre operation. In particular, we discuss GOS measures and requirements, and integer programming techniques for solving rostering problems. In Section 3 we explain how the cuts are generated, and present the RIIPS algorithm in more detail. Obviously, RIIPS is a heavy user of simulation, and so Section 4 discusses the use of efficiency improvement techniques to reduce the computational workload. Even a small simulation saving could make a large difference in run times.

In this paper we make no attempt to rigourously establish results, since our main goal is to convey the ideas underlying the approach. A forthcoming paper will describe the algorithm in more detail, and establish the required results.

To conclude this section, we would like to emphasize that the RIIPS approach is not limited to call centre staffing, although this is the application that motivated its development. Of course, this generality means that a great deal of work may be needed to tailor the approach to a given application, but nevertheless, the overall methodology remains the same.

2 BACKGROUND

In a typical call centre, a customer calls a number and, if a server is available, is connected to the server. The server then completes the service of the customer and concludes the call. If a server is unavailable, the customer is placed in a queue of calls until a server becomes available.

There are many factors that can complicate this deceivingly simple description of a call centre. For example:

- The arrival rate of calls varies throughout the day.
- Customers may balk if they are informed of their expected time till service.
- Customers may abandon (renege) if they are kept waiting too long.
- Not all customers are equal! Typically, one wishes to assign priorities to incoming calls, and answer high-priority calls first.
- Calls may be of multiple types, and therefore need to be assigned (if possible) to a particular subset of agents. For example, calls could be of a sales, technical support, or general enquiry nature. We would prefer the sales call to be handled by a sales agent, but if no sales agents are available, the call might be handled by a technical support agent.
- The number of available agents varies through the day due to staffing decisions, breaks, staff absence etc.

Call centre managers typically want to minimize operating costs, subject to the constraint of maintaining customer service. While operating costs are relatively easy to measure, the degree of customer service is not. From a customer's point of view, perhaps the major measure of service is waiting time in the queue before being connected to a server. We can model this using utility curves, reflecting the effect of different waiting times in the queue on the customer's satisfaction. These utility curves can vary from customer to customer, and indeed, with the customer's mood! Therefore, we choose a single utility curve that we believe is representative, and attempt to minimize some statistic associated with customer utilities. The utility curve is called a CGOS. Customers receive a CGOS corresponding to their waiting times.

Example: Customers may receive a CGOS for their waiting time (W) according to the following table.

Waiting Time (sec)	CGOS
$W = 0$	100
$0 < W \leq 20$	95
$20 < W \leq 60$	80
$60 < W \leq 300$	70
$300 < W$	50

If a customer abandons before receiving service, they might receive a 0 CGOS. Hence, in this example the CGOS is a function of whether the customer abandons, and the customer waiting time.

Various customer service requirements may be specified. For example:

- CGOS should exceed 50 for all customers.
- 95% of customers must receive a CGOS > 80.
- In any 2 hour window, the average customer CGOS should exceed 80.
- During peak times, the average CGOS should exceed 50, while at other times it should exceed 80.
- The expected CGOS for a customer arriving at any time throughout the day should exceed 80.

Because of the random nature of call centre operation, for any given day these requirements (with the exception of the last one) can only be met with a certain probability.

Our requirements on the GOS measure are that

1. customer service must be computable on a period by period basis.
2. customer service is monotone (increasing) and concave in the service agent allocations (the vector r in the introduction), and
3. any period of substandard customer service may be improved to a satisfactory level by adding service agents either during the substandard period, or in some period(s) before the substandard period begins (to reduce congestion at the start of the period).

In the introduction we mentioned a problem involving the processing of passengers arriving at an airport. In this problem (Mason, Ryan and Panton 1998), least-cost rosters are required that ensure 80% of the passengers on each aircraft will be processed within 60 minutes of the plane touching down. Each period i receives a GOS of either 0 (fail) or 1 (succeed), where a 1 is awarded unless some flight fails to meet its 80% target during the period.

The requirements listed above are sufficiently flexible to incorporate this staffing problem.

To conclude this section, we introduce an integer programming approach for solving rostering problems. Our intention is primarily to demonstrate that such an approach is feasible. For more detail than can be presented here, see Ryan (1992).

The first step is to create many "lines of work" (or tours of duty). A line of work is a pattern of work that an employee may follow during the rostering period. A line of work is represented as a column of 0's and 1's, with a 1 appearing in row j if the employee is available to serve customers in time period j. The lines of work may be generated to comply with union regulations and other restrictions on employee work patterns (lunch breaks etc.).

Let L be the matrix consisting of the line of work columns, and x_i be the number of employees following line i. Let c_i be the cost of the ith line of work. Let x and c be the vectors consisting of the x_i's and c_i's respectively. Let s be a vector of employee requirements (e.g., $s_1 = 3$ if at least 3 employees are required in period 1). In the two phase approach to solving these problems, s is a parameter that is predetermined, but we will treat it as a vector of variables. The required IP is therefore

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Lx - s \geq 0 \\ & As \geq b \\ & x, s \geq 0 \text{ and integer.} \end{array} \quad (1)$$

Observe that $r = Lx$ is the vector of the number of staff available in each of the periods. The constraints $Lx \geq s$ therefore ensure the roster maintains the minimum number required in each period. In the two phase approach to these problems we might take $A = I$, the identity matrix, and b to be the per-period service agent numbers required, i.e., we might restrict the variables s to be lower bounded by b. In the RIIPS approach, the structure of A and b is more complicated, and changes as the algorithm progresses. In particular, each row of A reflects a cut as determined by the simulation, so that as the algorithm progresses, rows will be added to the matrix A.

3 THE ALGORITHM

The RIIPS approach begins by obtaining a lower bound on the service requirements s in (1). This lower bound could be taken to be 0, but it is almost certainly more efficient to determine more reasonable lower bounds first. One way to achieve more reasonable lower bounds is as follows. Each period i is simulated assuming the system is empty at the start of the period. This is the "best" that one could hope for in terms of work passed over from the previous period. A number of simulations of the period are performed while adjusting the number of servers available in period i until the GOS in period i, Q_i, is "just" acceptable, i.e., we choose s_i so that Q_i is acceptable, but if s_i were one less, Q_i would be unacceptable. The presence of extra work at the beginning of period i can only degrade performance in period i, and so the value obtained is a lower bound.

Next, we solve the IP (1). This gives an initial roster, and from the roster, we can determine the number of service agents available during each period of the day $r = Lx$.

The vector r is now passed to the simulation. The simulation uses this vector of information to simulate the system and compute estimates of system performance over the periods of the day.

Recall that we are assuming that Q_i is a concave function of $r_1, \ldots, r_i$, so that we may write $Q_i = g_i(r)$, where g_i is a function only of $r_1, \ldots, r_i$. Further, the concavity of g_i implies that

$$Q_i(\tilde{r}) \leq g_i(r) + v_i^T(\tilde{r} - r), \quad (2)$$

where $\tilde{r}$ is an alternative to the vector r, and v_i is a supergradient of g_i at r. (We use the terminology "supergradient" as the analogue of a subgradient for convex functions; see Bazaraa, Sherali and Shetty 1993, p. 85 for subgradient definitions etc.) But we require that Q_i be above some minimum threshold m_i say, so that (2) gives

$$v_i^T \tilde{r} \geq v_i^T r + m_i - g_i(r),$$

or, in the notation of our IP,

$$v_i^T s \geq v_i^T r + m_i - g_i(r). \quad (3)$$

Observe that (3) defines a cut on the s variables of the IP. Since i was arbitrary, if we can obtain supergradients for each of the periods, we will obtain a cut for each of the periods. The IP may be augmented by some subset of these cuts (or possibly a combination of them), and resolved. The entire process then iterates (simulate—add cuts—simulate—add cuts) until some measure of convergence is achieved.

The RIIPS algorithm may be summarised as follows.

Algorithm 1 (RIIPS)

```
1.  Determine the initial lower bounds $b$ on $s$.
2.  Set $A = I$.
3.  Solve the rostering IP,
    obtaining $r = Lx$.
4.  Pass $r$ to the simulation and simulate to
    obtain the quality measures $Q_1, Q_2, \ldots, Q_p$.
5.  If convergence criteria are satisfied
6.      **Stop** and report $x$ and the $Q_i$'s.
7.  else
```

```
8.      Determine one or more cuts as above.
10.     Add the cut(s) to the rostering IP,
        i.e., augment $A$ and $b$.
11.     Go to Step 3.
12.  end if
```

The question remains as to how to obtain the supergradients v_i. These might be obtained using gradient estimation techniques such as the likelihood ratio method (Glynn and L'Ecuyer 1995) or Infinitesimal Perturbation Analysis (Fu and Hu 1997). However, because of the integer nature of the vector r, it is unclear whether these methods will provide appropriate results. Furthermore, these gradient estimation techniques are not applicable to all simulations. Certain regularity conditions must be met to ensure that the results are valid, and these conditions may prove difficult to verify. However, it is worth noting that these methods provide gradient estimates based on a *single* simulation run, and therefore are (relatively) low computational effort approaches. If they are applicable, and easily implemented, they should be seriously considered.

A more robust approach is to use finite differencing. The approach here would be to perturb r_i to $r_i + 1$ or $r_i - 1$, and then reperform the simulation. This will allow estimates of the ith component of the supergradient vectors v_j for each period $j \geq i$ to be obtained. (Note that for $j < i$, $v_j(i) = 0$, since the performance in period $j < i$ does not depend on r_i.)

If one desires gradient information for every period throughout the day, then this approach will require an additional number of simulations equal to the number of periods. However, if one only desires gradient information for the first k periods (say), then an additional k simulations will need to be performed. This observation may lead to computational savings in the simulation phase of the RIIPS algorithm. Further approaches to computational savings are discussed in the next section.

4 EFFICIENCY IMPROVEMENT

Algorithm 2 could, and probably will, require a very large amount of computation. Many IPs will need to be solved, and many simulations will be required before convergence is achieved. Therefore, any performance improvements that can be made at any stage will be most welcome. Because of the extensive use of simulation, one might consider the use of efficiency improvement techniques.

For large efficiency improvements, one must typically know and exploit a great deal of information about the stochastic system. Because we prefer to discuss our approach at a very general level (thereby maximizing its applicability), it is unlikely that the efficiency improvement techniques we advocate here will result in substantial improvements in performance. We will therefore adopt the philosophy that "every improvement helps", and discuss some "simple" approaches here. More sophisticated approaches might be developed for specific systems and performance measures.

The primary efficiency improvement technique we advocate is the use of common random numbers, in the following sense. For most performance measures, we will need to simulate several replications of a day's operation to obtain the Q_i's. Let us assume that we decide to simulate n days. While the RIIPS algorithm is running, the only thing that is changing (from the simulation's point of view) is the number of servers available at different times of the day. Therefore, we could use one set of n days worth of customer dependent data such as arrival times, service times, and abandonment times throughout the optimization process.

Note that there are two ways to implement this approach. The first would be to simply reset the random number seeds for the simulation before each replication. This has the advantage of allowing the use of pre-built simulations of the call centre, with *virtually no modification*. We also obtain the well-documented advantages of common random numbers when comparing stochastic systems; see, e.g., Chapter 11 of Law and Kelton 1991. (The stochastic systems we are comparing are those with different server allocations).

However, if we are willing to implement common random numbers in a different manner, further computational savings can be realized. Suppose the customer data is generated only once and stored. Then this data can be recalled when needed to compute sample paths. There are two additional advantages to this method for implementing common random numbers.

1. Since the data are generated only once, great computational savings are made in random variate generation. Of course, we will need to store and retrieve the simulated data, but the cost of this could be relatively minor when compared to generating the data many times.

2. One would expect that the server allocations in each period (the vector r) will not change tremendously from iteration to iteration. Since the simulation data are always the same, the sample paths will be identical up to the first period when server allocations differ. Hence, if we store the state of the system at the conclusion of each period, we may "warm-start" the current simulation from that point on. This could yield considerable savings when the algorithm begins to focus on rostering the end of the day.

Of course, these advantages come at the cost of having to develop special code to perform the simulation.

This is a significant disadvantage. The decision as to which approach to select (to implement common random numbers) is therefore application dependent.

The use of common random numbers as outlined here can basically be viewed as an application of the method of sample-path optimization (see Robinson 1996 for analysis and references). Therefore, we inherit the advantages and disadvantages of this optimization approach.

A second technique that one might consider is the use of antithetic variates. In this case, one could generate the first day's data, and then use an antithetic random number stream to generate the second day's data. This approach could be repeated for the third and fourth days, and so on. See Chapter 11 of Law and Kelton 1991 for further details on how to implement antithetic variates, and when they might have the greatest effect.

Observe that one can simultaneously use both of these techniques to obtain efficiency improvement.

5 IMPLEMENTATION

We do not yet have an implementation of RIIPS, but several implementation details are worth discussing at this point.

Perhaps the "cleanest" implementation would involve a simulation module, an integer programming module, and a controlling parent application.

The simulation module should be capable of running as a subroutine, so that parameters (the staff allocations) may be set by the parent application before executing the simulation, and results (the quality measures and/or gradients) may be returned to the parent application. This may be achieved using "off-the-shelf" simulation software by performing all communication between the parent application and the simulation module through a disk file. Although disk access is slow compared to "on-board" processing, the bulk of the effort will almost certainly be devoted to solving IPs and performing the simulation. A second approach to achieve this communication could be to employ "user inserts" (software hooks) as described in Boesel and Nelson (1998), although not all simulation software has this capability.

Similar comments regarding communication apply to the integer programming module.

ACKNOWLEDGMENTS

We would like to thank David Burgess of VoiceTech Ltd., and Richard Thomson for many stimulating discussions, from which these ideas developed.

REFERENCES

Bazaraa, M. S., H. D. Sherali, and C. M. Shetty. 1993. *Nonlinear Programming: Theory and Algorithms.* Wiley, New York.

Boesel, J., and B. L. Nelson. 1998. Accounting for randomness in heuristic simulation optimization. *Proceedings of the 12th European Simulation Multiconference*, ed. R. Zobel and D. Moeller, 634–638. Society for Computer Simulation International, Manchester, England.

Eitzen, G. E. 1994. *Telephone Resource Allocation Problem.* Honours Thesis, School of Mathematics, University of South Australia.

Fu, M., and J. Q. Hu. 1997. *Conditional Monte Carlo. Gradient Estimation and Optimization Applications.* Kluwer, Boston.

Glynn, P. W., and P. L'Ecuyer. 1995. Likelihood ratio gradient estimation for stochastic recursions. *Advances in Applied Probability* 27: 1019–1053.

Jennings O., A. Mandelbaum, W. Massey, and W. Whitt. 1996. Server staffing to meet time-varying demand. *Management Science* 42: 1383–1394.

Law, A. M., and W. D. Kelton. 1991. *Simulation Modeling and Analysis, 2nd ed.* McGraw-Hill.

Mason, A.M., D.M. Ryan, and D.M. Panton. 1998. Integrated simulation, heuristic and optimisation approaches to staff scheduling. *Operations Research* 46: 161–175.

Mehrotra, V. 1997. Ringing up big business. *OR/MS Today* 24(4).

Robinson, S. M. 1996. Analysis of sample-path optimization. *Mathematics of Operations Research* 21: 513–528.

Ryan, D. M. 1992. The solution of massive generalized set partitioning problems in aircrew rostering. *Journal of the Operational Research Society* 43: 459-467.

Thomson, R. (1998). *Decision Support for Call Centre Design and Management.* Master's Thesis, Department of Engineering Science, University of Auckland, New Zealand.

AUTHOR BIOGRAPHIES

SHANE G. HENDERSON joined the Industrial and Operations Engineering Department at the University of Michigan (Ann Arbor) in 1997 after completing his Ph.D. in Operations Research at Stanford University. He is currently a lecturer in the Department of Engineering Science at the University of Auckland, while on leave from the University of Michigan. His research interests include discrete-event simulation and queueing theory. He likes cats.

ANDREW J. MASON Andrew Mason is a lecturer in the Department of Engineering Science at the University of Auckland, New Zealand. His research interests include the development of optimisation systems for staff scheduling problems and the implementation of these within PC-based systems. His research activities are motivated by scheduling projects undertaken for a number of New Zealand companies and Government departments. He can be reached at a.mason@auckland.ac.nz or via www.esc.auckland.ac.nz/Mason.