Mathematical Institute

Shift scheduling with non-stationary stochastic demand

Candidate:

John Pougué Biyong

Lecturer:

December 2018

1 Introduction

The fast delivery market has been growing critically over the past years ([12]). Such service systems with limited opening hours face with stochastic and non-stationary[1] demand. In this context, determining staffing requirements is a challenge. However, up to date, shift scheduling with non-stationary stochastic demand has received limited attention in the academic literature. Much research has focused on scheduling problems in call centers (see, e.g, [1], [2] or [3]). In this paper, fast delivery shift scheduling problems with service-level constraints are formulated as mixed-integer programs and investigated through a simulation-based branch-and-bound approach. Section 2 specifies the problem formulation. Section 3 provides and discusses a technique to solve the problem. Section 4 applies the method to a real-world case study.

2 Problem Formulation

2.1 Context

Fast delivery systems must spread workers over predefined work shifts to satisfy demand. Demand is called *customer load*. Figure 1 displays the typical demand curve experienced by a food delivery system over a day. Time is cut into short periods called *staffing intervals* on which *capacity* (i.e number of workers) is assumed steady. Note that work shifts may overlap each other and be of varying durations. Shifts are the (longer) periods which workers are allocated to. Staffing intervals (or staffing periods) are simply the short time periods obtained by discretizing time (intervals of 5, 10, 15 min...).

Provided any customer load forecast, determining staffing requires to quantify the labor needed during each staffing interval to ensure satisfactory customer service and select staff shifts that cover this determined labor amount. The latter is solved through a set-covering integer program formulated in 2.2.

As proposed by [4] and [5], (i) customer load arrival is assumed to follow a Poisson process with time-dependent rate $\lambda(t)$. Quality of service (ii) is measured in terms of customer waiting time. More precisely, we require that the (virtual) waiting time W_t encountered by a customer arriving at time t and willing to wait infinitely satisfies:

[1] i.e. the demand rate varies over time

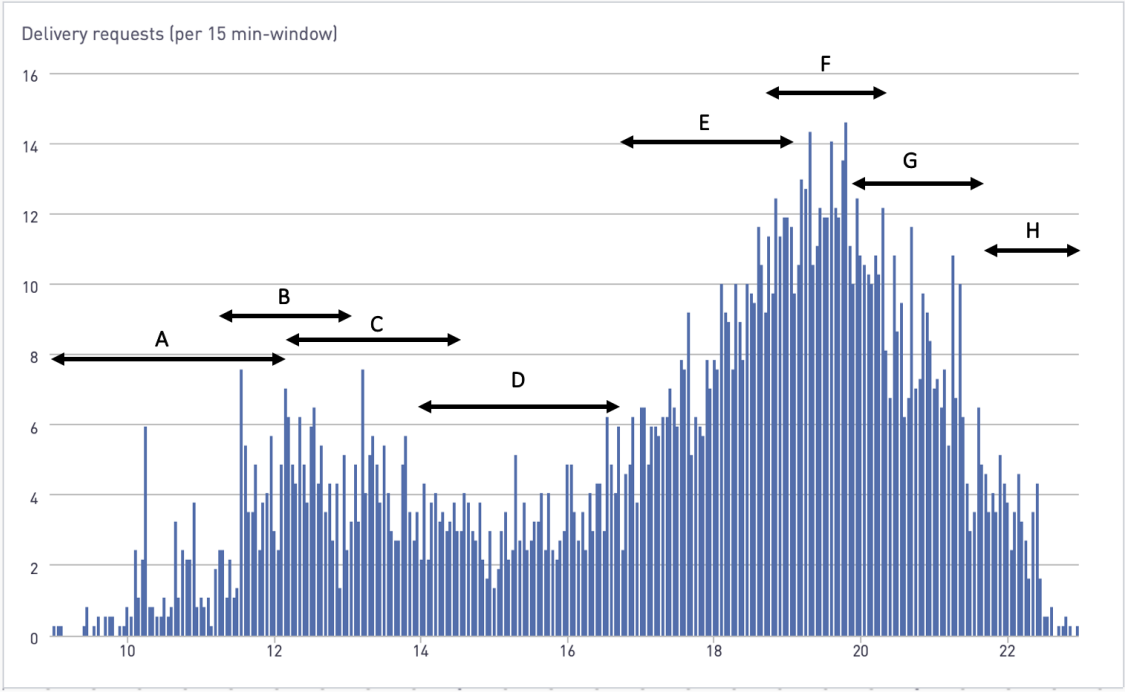

Figure 1: Schematic representation of time-varying customer load. In black, predefined work shits.

$$\mathbb{P}(W_t > \tau) \leq \alpha \quad \forall t \tag{1}$$

where τ is the maximum allowed waiting time and α the target probability of excessive waiting. Equation (1) is called *service level (or performance) constraint*. We define the staffing levels $s_1, ..., s_p$ as the numbers of workers required in the p staffing intervals. Note that the waiting time depends on the staffing levels, hence so does the constraint:

$$\mathbb{P}(W_t > \tau) = \mathbb{P}(W_t > \tau \mid \mathbf{s})$$

where $\mathbf{s} = (s_1, ..., s_p)$.

2.2 Problem statement

The staffing problem can be formulated as:

$$\begin{aligned} \min \quad & \mathbf{c}^\intercal \mathbf{w} = \sum_{j=1}^{n} c_j w_j \\ \text{s.t.} \quad & \mathbf{A}\mathbf{w} = \mathbf{s} \\ & \mathbf{w} \geq \mathbf{0} \qquad \text{and integer} \end{aligned} \tag{2}$$

$$\text{while} \quad \mathbb{P}(W_t > \tau \mid \mathbf{s}) \leq \alpha \quad \forall t$$

where

- $\mathbf{c}$ is the shift cost vector (and c_j is the cost of shift j, expressed in worker hours)
- $\mathbf{w} = (w_1, ..., w_n)^\intercal$ where w_j is the number of workers in shift j
- $\mathbf{s} = (s_1, ..., s_p)^\intercal$ where s_i is the number of workers in staffing period i
- $\mathbf{A} = (a_{i,j})_{ij}$ where $a_{i,j}$ is 1 if staffing period i is in shift j, 0 otherwise
- n is the number of shifts
- p is the number of staffing periods

In reality, any shift scheduling is admissible as long as it covers the required minimal staffing levels. Therefore, as proposed in [6], the constraint $\mathbf{Aw} = \mathbf{s}$ can be relaxed into $\mathbf{Aw} \geq \mathbf{s}$. Thus, for any given staffing vector $\mathbf{s} = (s_1, ..., s_p)^\intercal$, the shift covering problem can be formulated as:

$$\min \quad \sum_{j=1}^{n} c_j w_j \tag{3}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} a_{i,j} w_j \geq s_i \quad i = 1, ..., p \tag{4}$$

$$s_i \geq 0 \quad \text{and integer } i = 1, ..., n \tag{5}$$

$$\mathbb{P}(W_t > \tau \mid s_1, ..., s_n) \leq \alpha \quad \forall t \tag{6}$$

3 General Methodology

This section presents a method to solve the shift scheduling problem under stochastic non-stationary demand and with service-level constraints. It uses a branch-and-bound approach to decompose the problem into shift covering integer subproblems[2] (3)-(5) and simulation to check on the validity of constraint (6). Section 3.1 explains the construction and the exploration of the search tree. Section 3.2 describes how simulation is conducted. Section 3.3 provides a method to obtain the branch-and-bound algorithm inputs. Section 3.4 discusses the general method.

[2]The set covering problem was introduced by Dantzig (1954) [7].

3.1 A branch-and-bound algorithm

The method presented in this section is attributed to [5]. It requires three staffing vector inputs:

- an initial feasible solution $\mathbf{s^{init}}$. It is any staffing vector that leads to a shift vector satisfying the service level constraint. The corresponding shift vector is obtained as the solution of problem (3)-(5).
- a lower bound vector $\mathbf{s^{LB}}$ that contains a set of lower bounds on the staffing requirements for each staffing period $i = 1, ..., p$. We explain how to find $\mathbf{s^{LB}}$ in 3.3.
- an upper bound vector $\mathbf{s^{UB}}$ that contains a set of upper bounds on the staffing requirements for each staffing period $i = 1, ..., p$ (see 3.3).

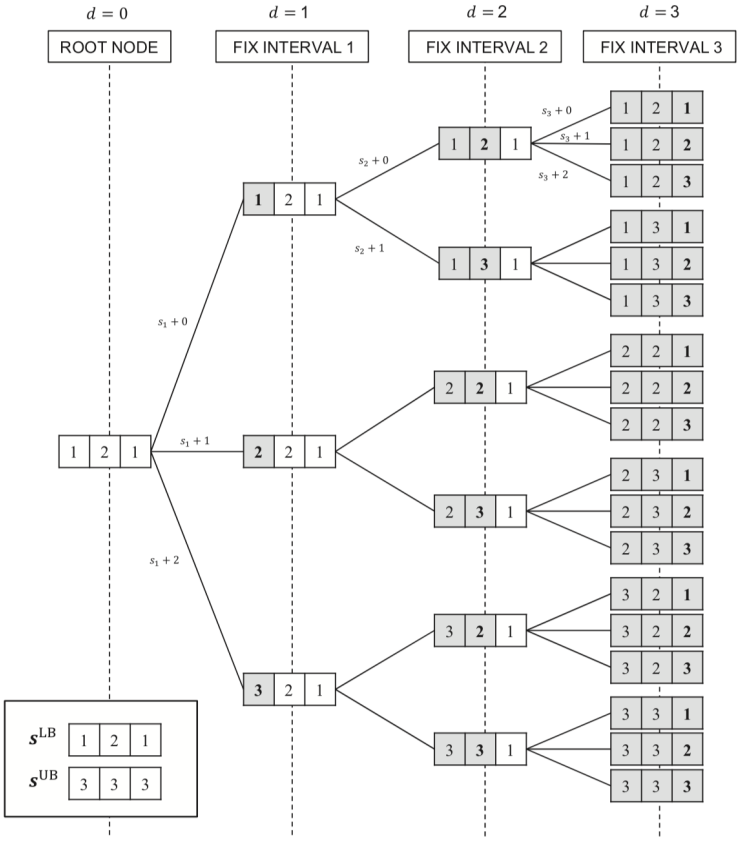

Figure 2: Example of tree structure. From [5].

Figure 2 presents an illustration of the tree structure, for p=3. Each node represents a staffing vector $\mathbf{s}$, with a corresponding staffing cost c_s (expressed in worker hours)

and a corresponding (3)-(5) problem. The tree is explored in a depth-first manner. The root node of the tree is $\mathbf{s^{LB}}$ and each level of the tree is denoted by its depth d which corresponds to the d-th staffing period. For any parent node of depth d, child nodes are generated by increasing the capacity in staffing interval d+1 (and leaving all the other capacities unchanged). Hence, the staffing level d+1 of the child nodes lies between the lower bound s_{d+1}^{LB} and the upper bound s_{d+1}^{UB}. Moreover, child nodes are explored in the increasing order of capacity c_s. With these elements in mind, the tree structure satisfies:

Theorem 1. *Let* $\mathbf{s}$ *be a node of depth* d *and* c_s *the corresponding staffing cost. The staffing cost of any child node (of depth* $\geq d$+1*) and any unexplored node of depth* d *is at least* c_s*.*

Proof. Recall that staffing costs are expressed in terms of worker hours. Any child node has the same staffing capacities as $\mathbf{s}$ for staffing intervals $\leq d$ and higher (or equal) staffing capacities on staffing intervals $> d$, so the overall staffing cost is higher or equal to c_s. Similarly, any unexplored node of depth d has the same capacities as $\mathbf{s}$ except on staffing interval d where it is strictly higher. □

Although all nodes are enumerated implicitly, they are not all explored explicitly. Indeed, for each of them, a problem (3)-(5) must be solved to find the corresponding shift vector and a simulation must be run to check on the performance criteria for this shift vector. As those two steps can be computationally expensive, rules are implemented to *fathom* nodes as soon as they fail at meeting specific requirements. A node is said to be fathomed if it is discarded from the search procedure, along with all its child nodes. Throughout the search tree procedure, the best feasible shift vector found so far is stored ($\mathbf{w}^*$ with shift cost c_w^*). At the start, $\mathbf{w}^*$ is initialized to $\mathbf{w^{init}}$ (the shift vector solution of the shift covering problem (3)-(5) solved with $\mathbf{s} = \mathbf{s^{init}}$). Any explored node (staffing vector) is assessed through 4 steps at most.

Step 1 *Evaluate staffing cost* c_s - the staffing cost c_s is the number of worker hours that corresponds to the staffing vector $\mathbf{s}$. If $c_s \geq c_{w^*}$, $\mathbf{s}$ can be fathomed along with all its child nodes and all unexplored nodes from the same parent node according to Theorem 1. The algorithm then proceeds to the next unexplored node in the tree: this can be a node at depth $d-1$ along the same branch as the parent node, or a node higher in the tree (if backtracking takes place). This rule is referred to as Fathom[$\mathbf{c_s}$].

Step 2 *Evaluate shift cost* c_w - Let c_w be the solution cost for integer program (3)-(5). If $\mathbf{s}$ has not been fathomed in Step 1, the LP relaxation of problem (3)-(5) is solved. According to Proposition 2.2 of lecture notes [8], the related shift cost c_w^{LP} satisfies $c_w \geq c_w^{LP}$. Hence, as before, if $c_w^{LP} \geq c_{w^*}$, $\mathbf{s}$ can be fathomed along with all its child nodes and all unexplored nodes from the same parent node and the algorithm then proceeds to the next unexplored node. This rule is referred to as Fathom[$\mathbf{c_w^{LP}}$]. Otherwise, problem (3)-(5) is solved with the integrality constraints included. Again, if $c_w \geq c_w^*$, the node $\mathbf{s}$ is fathomed along with its underlying nodes and unexplored nodes from the same parent. This rule is referred to as Fathom[$\mathbf{c_w}$].

Theorem 2. *Let* $\mathbf{w}$ *be the shift vector solution for a given staffing vector* $\mathbf{s}$*. Let us assume that* $\mathbf{w}$ *violates the performance constraint. Let* $t^e = min\{t \mid \mathbb{P}(W_t > \tau) > \alpha\}$ *the first time instant the constraint was violated. Let* i^e *denote the staffing interval that contains* $t^e + \tau$*. All* $\mathbf{s}'$ *for which* $s'_i \leq s_i$ *for all* $1 \leq i \leq i^e$ *are unfeasible as well, irrespective of the capacity in intervals* $i > i^e$*.*

Proof. To attain a waiting time $\leq \tau$, a customer arriving at t^e must enter the system at $t^e + \tau$ at the latest. If this condition is not satisfied for staffing vector $\mathbf{s}$, it cannot be satisfied neither for staffing vectors requiring fewer (or the same number of) workers ahead of time. □

Step 3 *Check if* $\mathbf{w}$ *was simulated before* - Different $\mathbf{s}$ vectors can lead to identical vectors. Therefore, it is possible that $\mathbf{w}$ has already been simulated at a previous node. To avoid running unnecessary expensive simulations, unfeasible simulated shift vectors are stored in a set $\mathbf{B}$. Thanks to Theorem 2, we can define a fathom rule termed Fathom[$\mathbf{i^e}$]. We distinguish three cases:

- $i^e < d$: the parent node at depth i^e can be fathomed along with all its child nodes and the next unexplored node in depth i^e is proceeded.
- $i^e > d$: we branch to the next unexplored node at depth i^e.
- $i^e = d$: the node at depth d can be fathomed and the algorithm proceeds with the next unexplored node at level d.

If the shift vector $\mathbf{w}$ was not simulated before, we go to step 4.

Step 4 *Evaluate* **w** *through simulation* - The simulation procedure is detailed in 3.2. If **w** is feasible, a new optimum has been found and $\mathbf{w}^*$ and c_{w^*} are updated. All the child nodes and the unexplored nodes sharing the same parent node can be fathomed as they cannot improve the optimum. This rule is named Fathom[$\mathbf{w}^*$]. As in steps 1 and 2, the algorithm proceeds with the unexplored nodes in the search tree. Otherwise, if **w** is unfeasible, the Fathom[$\mathbf{i^e}$] rule is applied and **w** is stored in **B**.

Appendix B presents the related pseudo-code proposed by [5]. We implement this code in Python.

3.2 Simulation

The planning horizon (typically a day) lasts T minutes and is divided into several staffing periods $i = 1, .., p$ which start at time t_i and end at t_{i+1} respectively. The staffing periods last Δ_p so that $\mathbf{t_p} = \{t_1, t_2, ..., t_p\} = \{0, \Delta_p, 2\Delta_p, ..., T - \Delta_p\}$.

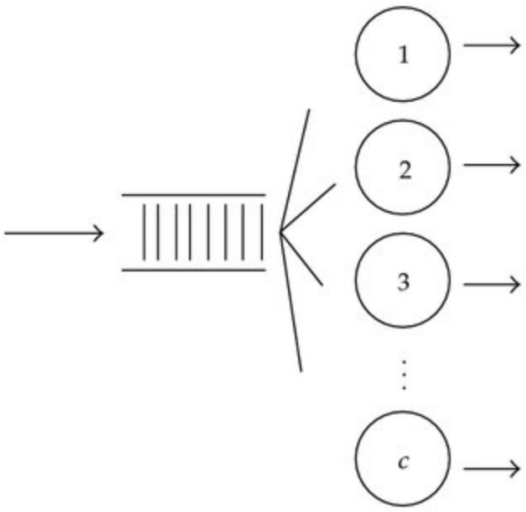

Figure 3: Representation of a first-come first-served queueing process. From [11].

We use a $M_t/M/c$ queuing model with first-come first-served discipline (see Figure 3): multiple servers serve jobs that arrive according to a $\lambda(t)$-rate Poisson process[3] and have exponentially distributed service times ([13]). For simplicity, we assume that the customers entering the system during staffing interval i all arrive at t_i. Performance is evaluated at $\Delta_p, 2\Delta_p, ..., T$ and customer abandonments that occur before τ are not

[3]Typically, $\lambda(t)$ is sinusoidal and mimics customer load trends seen in Figure 1

counted. We also assume that servers will work overtime at their shift end to finish the ongoing service instance if any.

For any shift vector $\mathbf{w}$, the queueing process is simulated repeatedly over the planning period. $\mathbf{w}$ is a satisfactory candidate if it violates the performance constraint less than α times (in proportion of the number of simulations). Otherwise, the first t_i such that $\mathbb{P}(W_{t_i} > \tau) > \alpha$ is stored in the set $\mathbf{B}$ of unfeasible. We implement a Python module provided in Appendix A. To the best of our knowledge, no simulation script algorithm taking this set of assumptions into consideration is provided in the literature, even though [9] presents a sophisticated mathematical approach to check on the performance constraint.

3.3 Inputs

3.4 Discussion - Suboptimality and efficiency

4 Case study

By using predictive techniques, it is claimed that demand rate $\lambda(t)$ follows a sinusoidal pattern fluctuating around the average λ_0:

$$\lambda(t) = \lambda_0(1 + R\cos(c_1 t + c_2))$$

where t expressed in minutes, R a variation amplitude factor.

5 Conclusion

References

[1] Athanassios N. Avramidis , Wyean Chan, Pierre L'Ecuyer (2009), *Staffing multi-skill call centers via search methods and a performance approximation*, IIE Transactions, 41:6, 483-497.

[2] Stefan Helber, Kirsten Henken (2010) *Profit-oriented shift scheduling of inbound contact centers with skills-based routing, impatient customers, and retrials*, OR Spectrum, Volume 32, Issue 1, p 109–134.

[3] Júlíus Atlason, Marina A. Epelman, Shane G. Henderson (2007), *Optimizing Call Center Staffing Using Simulation and Analytic Center Cutting-Plane Methods*, Management Science, Vol. 54, No 2.

[4] Armann Ingolfsson, Fernanda Campello, Xudong Wu, Edgar Cabral (2010), *Combining integer programming and the randomization method to schedule employees*, European Journal of Operational Research, Vol. 202, Issue 1, p 153-163.

[5] Mieke Defraeye, InnekeVan Nieuwenhuyse (2016) *A branch-and-bound algorithm for shift scheduling with stochastic nonstationary demand*, Computers and Operations Research, Vol. 65, p 149-162.

[6] Mehmet Tolga Cezik, Pierre L'Ecuyer (2008) *Staffing Multiskill Call Centers via Linear Programming and Simulation*, Management Science, Vol. 54, No 2.

[7] George B. Dantzig, (1954) *A Comment on Edie's "Traffic Delays at Toll Booths"*., Journal of the Operations Research Society of America 2(3):339-341.

[8] Raphael Hauser *Integer Programming Notes*, Lecture Notes for B6.3 Integer Programming module at University of Oxford, MT 2018-2019.

[9] Linda V. Green, João Soares (2007) *Computing Time-Dependent Waiting Time Probabilities in M(t)/M/s(t) Queuing Systems*, Manufacturing and service operations management, Vol. 9, No 1.

[10] Mieke Defraeye, InnekeVan Nieuwenhuyse (2013) *Controlling Excessive Waiting Times in Emergency Departments: An Extension of the ISA Algorithm*, Available at SSRN: https://ssrn.com/abstract=1967004 or http://dx.doi.org/10.2139/ssrn.1967004

[11] Frederico Samartini et al. (2011) *Upper Bounds on Performance Measures of Heterogeneous M/M/c Queues*, Mathematical Problems in Engineering, Vol. 2011.

[12] *https://www.mckinsey.com/industries/travel-transport-and-logistics/our-insights/how-will-same-day-and-on-demand-delivery-evolve-in-urban-markets*, McKinsey and co., 2017.

[13] *https* : *//en.wikipedia.org/wiki/Queueing*$_t$*heory*, Queueing theory. See "Kendall's notation" part.

[14] *https* : *//www.youtube.com/watch?v* = $Ww69GUEIgVI$, Staffomatic, Shift Planner Deliveroo.

A Simulation module

B Branch-and-bound pseudocode

```
function ExploreTree()

    Set initial feasible staffing vector s^init
    Determine w^init

    w* ← w^init    // Initialize best feasible solution so far
    s̄_w* ← s̄_w^init
    c̃_w* ← c̃_w^init

    Set bounds s^LB and s^UB

    s ← s^LB    // Initialize root node
    d ← 0    // Initialize depth

    repeat
        Determine c_s
        if c_s ≥ c̃_w* then
            Fathom[c_s]:
            s_d ← s_d^LB    // Restore capacity in interval d
            d ← d − 1    // Go to parent node.
            Backtrack()
        else
            Solve Problem (2-4) to obtain {w, c̃_w, s_w}
            if c̃_w ≥ c̃_w* then
                Fathom[c̃_w]:
                s_d ← s_d^LB    // Restore capacity in interval d
                d ← d − 1    // Go to parent node.
                Backtrack()
            else
                Check if w has been simulated before
                if w ∈ B :  then
                    i^e ← i^e of previously simulated solution
                    if d < i^e then
                        Branch(i^e)    // Proceed to child node on level i^e
                    else  //  d ≥ i^e so branching to underlying nodes is useless
                        while (d > i^e) do
                            s_d ← s_d^LB    // Restore capacity in interval d
                            d ← d − 1    // Go 1 level back
                        end while
                        Backtrack()
                    end if
                else
                    Simulate s_w to obtain i^e
                    if Feasible then
                        w* ← w    // Update best solution so far
                        s*_w ← s_w
                        widetildec_w* ← widetildec_w

                        Fathom[w*]:
                        s_d ← s_d^LB    // Restore capacity in interval d
                        d ← d − 1    // Go to parent node.
                        Backtrack()
                    else  // Infeasible solution
                        if d < i^e then
                            Branch(i^e)    // Proceed to child node on level i^e
                        else  //  d ≥ i^e so branching to underlying nodes is useless
                            Fathom[i^e] :
                            while (d > i^e) do
                                s_d ← s_d^LB    // Restore capacity in interval d
                                d ← d − 1    // Go 1 level back
                            end while
                            Backtrack()
                        end if
                    end if
                end if
            end if
        end if
    until (d = 0)
end function
```

Figure 4: Branch-and-bound pseudocode. See Figure 8 for side functions. From [5]

```
function BACKTRACK()   // Return to previous level and proceed
    with next node
  if d ≠ 0 then
    while (s_d = s_d^UB and d > 0 do
      s_d ← s_d^LB   // Restore capacity in interval d
      d ← d − 1   // Go 1 level back
    end while
    if d ≠ 0 then
      s_d ← s_d + 1   // Proceed to next node on the same level
    end if
  end if
end function
```

```
function BRANCH(d')   // branch to child node on level d'
  d ← d'   // Increment depth
  if s_d < s_d^UB then
    s_d ← s_d + 1   // Augment capacity in interval d
  else
    Backtrack()
  end if
end function
```

Figure 5: Side functions. From [5].