Tel-Aviv University

The Raymond and Beverly Sackler

Faculty of Exact Sciences

School of Mathematical Sciences

Mining of Routing Protocols in Queueing Systems with Applications to Call Centers

Thesis submitted in partial fulfilment of the requirements

for the M.Sc. degree of Tel-Aviv University

by

Pablo Liberman

This thesis was prepared under the supervision of

Prof. Isaac Meilijson and Prof. Avishai Mandelbaum

February 2017

Acknowledgement

This research thesis was written under the supervision of Professor Isaac Meilijson, from the Faculty of Exact Sciences in Tel-Aviv University, and Professor Avisai Mandelbaum, from the Faculty of Industrial Engineering and Management in the Technion. I would like to thank them for the invaluable guidance and support throughout every stage of this work. During this process was notable their devotion to enriching me with their knowledge and experience.

I would like to thank the team in the SEELab from the Technion for their support during the empirical analysis in this work. They guided me on the use of the lab datasets and the lab SEEstat software for data analysis. Their support was crucial for the implementation of the models proposed in this work.

Also I would like to thank my dear family for supporting and encouraging me all the way through. Special thanks to my beloved Yamila, who strengthened me with her love, and to my sons Itai and Harel.

Finally, the generous financial help of the Tel-Aviv University Department of Statistics and Operations Research is gratefully acknowledged.

Abstract

In modern call centers, it is common to have multiple classes of customers and multiple agent pools. In these systems, a Skills-Based Routing (SBR) protocol matches classes and pools: it defines whether and to whom calls are routed (agents are assigned).

Call centers consisting of multiple customer classes and agent pools manage complex SBR protocols, commonly consisting of a large set of routing rules that evolve over time in a trial and error fashion. In this situation, two problems arise: 1. It is hard to maintain a reliable documentation of all the existing rules, and 2. Even when there exists adequate documentation of the protocols, the inter-dependencies between rules and their influence on routings are not easily understandable.

In this context, we argue that SBR protocol models can be extracted from routing logs that are recorded during system operations. Such models may guide operational management of SBR protocols.

The main purpose of this research is then to develop a method to mine routing protocols in queueing systems. We call our method **SBR-Mining**. In order to achieve our goal, we introduce a general model for SBR systems, a mathematical formulation of routing protocols, a routing data model, a definition of protocol goodness-of-fit measures and finally, a learning method.

To validate our method, we apply it to three case studies, and show how SBR-Mining is used to build routing models with more than 90% accuracy, under the proposed goodness-of-fit measure.

Our SBR-Mining method is described for a general model of routings in a network. Therefore, applications of the introduced method can be extended to general Process Mining studies, in order to discover routing protocols of processes in different business areas.

Contents

List of Figures

List of Tables

List of Acronyms

ACD . . . . . Automatic Call Distributors software

FIFO . . . . . First In - First Out.

FQR . . . . . Fixed Queue Ratio

IVR . . . . . Interactive Voice Response

LPF . . . . . Least Patient First

NIIT . . . . . No Induced Idle Time

PSS . . . . . Parallel Service Systems

QIR . . . . . Queue-and-Idleness-Ratio

QoS . . . . . Quality of Service

SBR . . . . . Skills Based Routing

SEE . . . . . Service Enterprise Engineering

VRU . . . . . Voice Response Unit

Chapter 1

Introduction

During the last century and the beginning of the current one, the service sector has grown to such an extent that it now exceeds 70% of the economic activities in Western countries. The service sector covers a wide spectrum of activities, e.g., education, professional services, healthcare services, financial services and government services.

In this thesis we focus mainly on telephone call centers, very commonly used by companies and organizations to communicate with their customers. Indeed, for many companies, such as airlines, hotels, retail banks and telecommunication companies, call centers provide the primary link between customer and service provider. In general, call centers have become a vital part of our service-driven society. As a result, call centers have also become an object for academic research. For an extensive review on call centers research, readers are referred to [7] and [1].

In modern call centers, it is common to have multiple classes of customers and multiple agent pools. The customer classes are differentiated according to their service needs. The agent pools are characterized by their skills, namely by the subset of customer classes that they can adequately serve and the quality of service that they can devote to each such class (skills). An important example of such large scale service systems are multi-skill call/contact centers. Such centers are often characterized by multiple classes of calls (classified according to type or level of service requested, language spoken, perceived value of customers, etc.).

When dealing with the operational management of multi-skill call/contact centers, one of the main issues to address is defining a Skills-Based Routing (SBR)

protocol: a set of rules that define whether and to whom calls are routed. These protocols are usually defined with the objective of minimizing system costs, such as workforce costs, while providing targeted quality of service (QoS) levels.

Call centers consisting of several customer classes and agent pools manage complex SBR protocols aimed to prioritize and route customers according to their classes, the load in the system and the individual customer service history.

In practice, multi-skill call/contact centers define a large set of routing rules that evolve over time in a trial and error form. These rules are programmed in advance in the automatic call distributors (ACD), and updated when a new QoS target is introduced or when an existing protocol fails to achieve one of the existing targets. In this context, two problems arise: 1. It is hard to maintain a reliable documentation of all the existing rules, and 2. Even when there exists adequate documentation of the protocols, the inter-dependencies between rules and their influence on routings are not easily understandable.

In this context, we introduce an **SBR-Mining** method, which deduces skill-based routing protocols from telephone calls data. The proposed method aims to extract from so-called "event logs", originated by the ACD, the rules that characterize system routings.

Extraction of information from process logs is widely studied in the Process Mining research area. Process Mining aims at developing techniques to extract business process information from their realization logs [32, 33]. The SBR-Mining method introduced in this work is hence a Process Mining technique, aimed at learning the protocol that controls process flow.

1.1 Contribution and Structure of the Thesis

Chapter 2 provides theoretical background and surveys some related literature. Section 2.1 introduces the management levels of queueing systems' operation: We describe the role of routing protocols in operations management and their relation to other management decisions. Section 2.2 surveys routing protocols studied in the literature in order to identify general structures of protocols to be afterwards used in our SBR-Mining method. Chapter 2 concludes with Section 2.3 that surveys the Process Mining literature related to protocols learning. We start with an introduction to Process Mining research, then survey previous applications of Pro-

cess Mining techniques that study service processes and finally survey the Process Mining literature that addresses the protocol learning question.

Chapter 3 presents our SBR-Mining method. We introduce a model to describe the system classes and pools, then formally define routing protocols in SBR systems and finally describe our learning method.

The SBR-Mining model introduced in Section 3.1 represents the call center as a network, where customer *classes* and agent *pools* are the nodes, and bipartite edges connect between agent pool nodes and the classes that the agents in the pools are capable of serving. In this model, the agent and customer nodes share the same characteristics in the sense that choosing a waiting customer to be served by an agent that becomes idle is similar to choosing an idle agent to serve an arriving call/customer.

A key property of the SBR-Mining network is that it does not adopt the common assumption in SBR literature that states that calls in the same class and agents in the same pool are statistically identical. In our network, nodes of customers or agents with different characteristics may exist.

In our SBR-Mining model the routing protocols are defined as functions of the system state, such as the load, characteristics of waiting customers and number of available agents. This definition distinguishes between two kinds of selection considered by the protocol to route a call: the first one is the selection of the agent pool that will serve the call and the second one is the selection of the agent in the selected pool that will serve the call. Both decisions exist also in the dual problem of agent selection: the first selects the customer class to be served by an available agent, and the second the waiting customer from the selected class that is served.

Section 3.2 presents the SBR-Mining method, which applies machine learning algorithms to learn the routing protocols. The method defines each routing rule of each node in the model as a classification problem. Then, applying learning algorithms the routing protocols are explored.

Following the SBR-Mining method description in 3.2, Chapter 4 illustrates the application of the method to two real-world databases. The chapter begins with a description of two real-world study cases and discusses some practical issues in SBR-Mining applications.

In Section 4.3, we describe some SBR-Mining examples. The first example applies our method to learn how calls from the same class are prioritized; the second

one learns the prioritization of available agents from the same pool, and the last example illustrates the learning of a protocol that selects which class will be served first by a specific agent pool. In these examples, we show how our SBR-Mining method estimates routing protocols that coincide in more than 90% of the routing samples.

The thesis ends in Chapter 5 with conclusions and suggestions for future research.

Chapter 2

Theoretical Background and Literature Review

2.1 Operational Levels of Service Systems

The operations management of a service system is here divided into three levels: system design, staffing and control [7, 10, 13]:

- Design: The long-term problem of determining the rules that partition customers into classes, and servers into pools; this typically includes overlapping skills (i.e. servers that can cater to more than one class of customers, and customers that can be served by several server pools).
  It is important to note that, in addition to call content and agent training, call classes and agent skills may be defined according to any of a wide set of attributes. Examples include operational attributes, such as the forecasted duration of service, and economic attributes, such as by how much an agent is compensated.

  Figure 2.1 introduces some canonical designs that are studied in the literature. For example, the N-design in the left is composed of two customer classes (queues), and two agent pools (circles). The lines connecting queues to agent pools indicate the pools eligible to serve each queue. In the N-design, the left pool is trained to serve only the left queue and the right pool is trained to serve both queues.

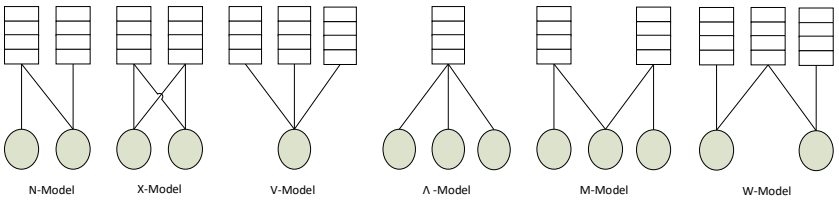

Figure 2.1: Canonical examples of service systems design

- Staffing: The short-term problem of determining how many servers are needed of each type in order to accommodate a given demand. These server types may be of overlapping skills.

- Control: The on-line problem of matching customer calls with appropriate servers. System controls are SBR rules for matching waiting calls with idle servers at event epochs, namely when a call arrives to the system (choose a server) or completes service (choose a customer).

An optimal operational policy defines a system design, a staffing level and a control protocol that minimizes the operational costs (usually workforce related costs) subject to Quality-of-Service (QoS) targets. These three management decisions are all interrelated. For example, the control protocol is constrained by the agent pools' skills defined at the design level. Therefore, these problems should be discussed in conjunction with one another. Yet, because of the complexity involved in addressing all these three levels jointly, they are typically addressed hierarchically and unilaterally in the literature [7].

2.2 Control in Service Systems (Skill-Based Routing) [1]

This section reviews SBR protocols that are studied in the literature. The protocols here provide the basis for SBR-Mining in subsequent chapters.

[1] This review is adapted from Gurvich, Liberman and Mandelbaum [15], with the authors' approval.

Theoretical Model

SBR systems are modeled in the literature as Parallel Service Systems (PSS). In a PSS model a queue is formed for each customer class and customers leave either by abandoning the queue or after service completion.

The model is described by a fixed set of customer classes $\mathcal{I} = \{1, \ldots, I\}$, a set of agent pools $\mathcal{J} = \{1, \ldots, J\}$ and the skill sets of each agent pool $I(j) \subseteq \mathcal{I}$. Similarly to agent skill sets notation, let $J(i) \subseteq \mathcal{J}$ be the set of agent pools that can serve i-class customers.

The SBR literature commonly assumes that customers from a common customer class are homogeneous (statistically identical) and independent, and similarly for agents within the same service pool.

Routing Protocols

Routing protocols are usually defined by two types of rules: (1) calls routing, namely whenever a service ends and there are queued customers, which customer (if any) should be routed to the server just freed, and (2) agent routing, namely whenever a customer arrives and there are idle servers, to which (if any) should the arriving customer be routed.

In the context of call centers, routing protocols are usually non-preemptive and non-anticipative. Non-preemption means that a customer service may not be interrupted before its completion with the intention of resuming it at a later time. Non-anticipation corresponds to the router having at its disposal only information about the evolution of the system up to the decision time.

Research of routing protocols focuses on finding a protocol that minimizes operational costs under a given system design and staffing. In the following sections, we review studied SBR protocols, which are grouped into two families: dynamic index rules and threshold reservation rules.

Dynamic Index Rules

Dynamic index rules originate in the Dynamic Allocation Indices introduced by Gittins [11]. Gittins describes a multi-armed bandit problem where arms may be pulled repeatedly in any order, and the result of a pull can be used to influence the next arm selection. In this model, arms are modeled by independent Markov

Decision Processes and the current state of a process is used to compute the reward that can be achieved by the process evolving from that state on, namely, by pulling an arm in the next step. The Dynamic Index policy consists of choosing at any time the arm stochastic process with currently the highest index value.

In SBR index rules each call's queue and each agent's pool is viewed as a "bandit arm", namely as an independent Markov Decision Process. The index routing protocol is defined by the reward function of selecting each queue or pool.

A special feature of index rules is that, at each decision epoch, the protocol chooses the customer class (or the agent pool) with the highest index, where the calculation of the index requires only **local** information. Namely, to compute a queue index, only state information that is needed for that queue. Similarly, an agent pool index depends only on the pool state.

Existing work on index-type protocols capture a queue state through the number of calls in the queue and the waiting time of the head-of-line call. Equivalently, an agent pool state is captured by the number of idle agents in the pool and the idle time of the longest idle agent in the pool.

To denote index rules, let $Q(t) = (Q_1(t), \ldots, Q_I(t))$ be the queue length vector at time t. Let $I(t) = (I_1(t), \ldots, I_J(t))$ be the idleness vector, namely, $I_j(t)$ is the number of idle agents in pool j at time t. Also, let $W^h(t) = (W_1^h(t), \ldots, W_I^h(t))$ be the head-of-the-line waiting-time vector: $W_i^h(t)$ is the waiting time of the customer that is at the head of the class-i queue at time t (i.e, the customer that has waited the longest). Finally, let $T_j^I(t)$ be the idle time (since last being busy) of the longest idle agent in pool j.

An index rule is then characterized by functions $(f_i,\ i \in \mathcal{I},\ j \in \mathcal{J})$ and $(g_j,\ i \in \mathcal{I},\ j \in \mathcal{J})$ so that, upon a call arrival, a class-i call is routed to an agent in pool

$$j \in \underset{l \in J(i) \bigcap \{Id_j(t) > 0\}}{\operatorname{argmax}} g_j(I_l(t), T_l^I(t)).$$

Upon service completion, a pool j agent will serve a customer from class

$$i \in \underset{k \in I(j) \bigcap \{Q_i(t) > 0\}}{\operatorname{argmax}} f_i(Q_k(t), W_k^h(t)).$$

Various index rules were studied in the SBR literature. We now introduce some of these rules:

- $c\mu$ **Rule:** A simple index rule is defined by the $c_i\mu_i$ index, where c_i denotes the waiting cost per unit of time in class i and μ_i denotes the class i calls service rate. This static index rule, known as the $c\mu$ rule, is shown to be optimal for minimizing waiting costs in many settings where delay costs are linear [29].

  The $Gc\mu$ rule (generalized $c\mu$) extends the $c\mu$ rules beyond linear delay costs, specifically to convex cost functions [2, 17, 23, 29, 34]. This rule is defined as follows:

$$f_{i,j}(Q_i(t), W_i^h(t)) = C_i'(W_i^h(t)),\ i \in \mathcal{I},\ j \in \mathcal{J},$$

  where $C_i(\cdot)$ is a convex cost function for the class-i waiting time and $C_i'(\cdot)$ is its derivative. Note that the $c\mu$ and $Gc\mu$ protocols define the calls selection rule $f_{i,j}()$, but not the agents selection $g_{i,j}()$: These protocols are usually studied in the conventional heavy-traffic regime in which there are asymptotically no idle servers[2]

  In [23] it is shown that the $Gc\mu$ rule minimizes both instantaneous and cumulative queueing costs over essentially all scheduling disciplines, preemptive or non-preemptive, in the heavy-traffic regime. In this work, $C_i(\cdot)$ is an increasing convex function for which $C_i'(0+) = 0$ (Further properties of $C_i(\cdot)$ are listed in the original work).

- **Static Index Rules:**

  In a static index rule, the index values of the queues and the pools are predefined constants. The call-selection problem is solved by assigning fixed priorities to call classes: an available agent is assigned to the highest-priority call that the agent is qualified to handle. Similarly, for each call class there exists an ordered list of qualified agent pools, and arriving calls are assigned to the first pool in the list that has an available agent.

  A prevalent static rule is specified through two non-negative matrices with integer entries: an $I \times J$ matrix $M^{c\to s}$ and a $J \times I$ matrix $M^{s\to c}$. The entry

[2] Efficient operation in large call centers leads to maintain high levels of agents utilization, which renders the function $g_j()$ to route arriving calls to idle agents immaterial. Operational regimes of call centers that focus on high utilization levels are called in the literature *efficiency-driven* [9].

$M_{ij}^{c\to s}$ specifies the priority of pool j for class i so that a class-i customer is routed upon arrival to the pool j that satisfies

$$j \in \underset{j\in J(i):I_j(t)>0}{\operatorname{argmax}} M_{ij}^{c\to s}.$$

If all servers in $J(i)$ are busy, the customer waits in the class-i queue. Similarly, $M_{ij}^{s\to c}$ specifies the priority of class i for pool j, so that when pool-j agents become available they serve the customer from class i with

$$i \in \underset{i\in I(j):Q_i(t)>0}{\operatorname{argmax}} M_{ij}^{s\to c}.$$

If all queues in $I(j)$ are empty, then these agents remain idle.

Note that the $c\mu$ index rule introduced before is an example of a prevalent static rule.

- **Dynamic Ratio Rules:**

  Ratio rules aim to keep a predefined balance (distribution) of the total system load among its queues. For that, a ratio rule routes an available agent to serve a customer from the queue of the class whose queue load exceeds by the most a predefined proportion of the system load. Similarly, to keep a predefined balance of the system servers' idleness among its pools, a ratio rule routes an arrival call to be served by an agent from the pool whose idleness exceeds by the most a specified proportion of the total idleness.

  Gurvich and Whitt introduce in [16] the Queue-and-Idleness-Ratio (QIR) rules. QIR rules are defined by two ratio functions $r(\cdot) = (r_1(\cdot), ..., r_I(\cdot))$ and $v(\cdot) = (v_1(\cdot), ..., v_J(\cdot))$, whose values define the desired load distribution among queues and pools respectively. $r(\cdot)$ and $v(\cdot)$ values are non-negative and each vector adds up to 1. Given two admissible state-dependent ratio functions r and v, QIR is defined as follows.

  - A newly available agent of pool j next serves the customer from the head of the queue of the class (from among those it is eligible to serve) whose queue length exceeds by the most the specified proportion of the total queue length:

$$i \in \underset{k\in S=I(j)\bigcap\{Q_i(t)>0\}}{\operatorname{argmax}} Q_k(t) - Q_S(t)\cdot r_k(Q_S(t)).$$

Here $Q_S(t) = \sum_{k \in S} Q_i(t)$.

- Similarly, an arriving customer of class i is routed to an agent of the pool whose number of idle agents exceeds by the most a specified state-dependent proportion of the total idleness.

$$j \in \underset{l \in S = \{J(i) \bigcap \{I_j(t) > 0\}\}}{\operatorname{argmax}} I_l(t) - I_S(t) \cdot v_l(I_S(t)).$$

Here $I_S(t) = \sum_{l \in S} I_i(t)$

QIR Ratio rules constitute an extension of index rules. While index values are computed using only queue or pool local data, ratio rules compute the index value using local and aggregate system data.

In [18] Gurvich and Whitt introduce the fixed-queue-ratio (FQR) routing rule, a special case of QIR where the predefined ratio functions (r and v) are fixed vectors. In that paper, the authors propose FQR routing protocols to satisfy diverse Quality of Service (QoS) constraints minimizing labor-related costs. For example, to satisfy a QoS constraint of the form "the proportion of customers of class i that wait more than x seconds must not exceed y_i" the optimal routing protocol is an FQR rule whose ratio function v is defined by the target proportions of each queue, y_i, multiplied to their corresponding arrival rate λ_i:

$$r = (\frac{\lambda_1 \cdot y_1}{\sum_k \lambda_k \cdot y_k}, ..., \frac{\lambda_I \cdot y_I}{\sum_k \lambda_k \cdot y_k})$$

Similarly to $c\mu$ Rules, FQR rules are studied in the heavy-traffic regime, in which the agent selection ratio function v is immaterial.

QIR rules, as introduced above, balance queue loads by setting queue lengths to predefined proportions. An alternative ratio rule is the Waiting-and-Idleness ratio (WIR), whose load balance is based on the average waiting time of the customers in each queue. In [17], Gurvich and Whitt show that, under specific settings, the QIR rule is asymptotically optimal to minimizing holding costs and the WIR rule is asymptotically optimal to minimizing delay costs.

Threshold Reservation Rules

Threshold reservation policies specify system states in which calls are not assigned to idle agents qualified to serve them. In these states, the protocol reserves agents

to serve future arriving calls from other classes without waiting. Threshold policies define when agents and calls are prevented from being routed, but do not specify how agents or calls are routed when the threshold conditions enable them to be routed. Thus, a complete SBR protocol must be complemented with additional routing specifications. To formalize the above, let us introduce a threshold reservation policy combined with an index-based selection.

Given non-increasing non-negative integer valued functions $K_{ij}(Q_i(t), W_i^h(t))$, $i \in \mathcal{I},\ j \in \mathcal{J}$, the agent reservation rule induces the following actions upon customer arrival and service completions:

- Upon arrival at time t, customers of class i are routed to an agent in pool

$$j \in \underset{J(i) \bigcap \{l: I_l(t) > K_{il}(0,0)\}}{\operatorname{argmax}} g_l(I_l(t), T_l^I(t)).$$

  If no such j exists, the customer waits in the class-i queue.

- Upon service completion in pool j, the newly available agent serves the customer from the head of the class-i queue, where

$$i \in \underset{l \in I(j) \bigcap \{l \in \mathcal{I}: Q_l(t) > 0 \wedge I_j(t) > K_{lj}(Q_k(t), W_k^h(t))\}}{\operatorname{argmax}} f_k(Q_k(t), W_k^h(t))$$

  If no such queue exists, the agent remains idle.

Thus, class-i customers are routed upon arrival to a type j agent only if the number of idle servers in pool j exceeds $K_{ij}(0,0)$. Similarly, type-j agents that become available serves the customers at the head of the class-i queue only if the number of idle servers in pool j exceeds a function of this customer's waiting time or length of queue.

Gurvich et al. [14] analyze a state-independent threshold policy for the multi-class single-pool model (a V-design) and shows that this policy asymptotically minimizes holding costs under delay constraints. Queue-length based thresholds were studied in [3], for an N-design system. Such reservation policies are also analyzed in [8] and [26].

Protocol Review Summary

The previous review introduced the most commonly studied protocol types in the literature. All the protocols introduced are described by the functions f_i, g_j and

K_{ij}, $(i \in \mathcal{I}, j \in \mathcal{J})$: f_i and g_j define the dynamic index value of each queue i and pool j; k_{ij} is the threshold function that defines system states in which an arriving call is not served even if there are idle agents capable of serving it.

We remark that restrictions on the original index functions of using only local information of the queue state (or pool state) can be relaxed to permit the use of non-local information (for example, the total number of waiting customers, or the average waiting time of recently-served customers).

The SBR protocols described here set a FIFO (first in - first out) regime selecting calls from the same class and selecting idle agents from the same pool. This setting is natural under homogeneous customer and agent assumption. Alternatives to this setting are personalized-queue regimes, where data on individual customers or agents is used to prioritize waiting calls and idle agents.

In [22], Mandelbaum and Momčilović define the so-called Least Patient First regime (LPF), where the estimated patience of waiting customers is used to select the next customer to be routed. In the LPF regime, the customer with the least estimated time to abandon is served first. In [22] the authors compare the LPF regime against a FIFO regime, and show that LPF can provide significant lower abandon rates over FIFO when the durations of overloaded periods are comparable to (im)patience times.

A personalized regime to route agents is introduced by Mandelbaum and Momčilović in [21]. In this case, agents that exhibit shorter service times get a higher preference to serve the next arriving call.

In the next chapter we introduce a routing model combining protocols to select classes and pools (SBR protocols), and protocols to select same-class calls and same-pool agents.

2.3 SBR-Mining as a Process Mining Study

Our SBR-Mining work could be viewed as a research in Process Mining, which studies business processes by extracting knowledge from their event logs, commonly available in today's (information) systems.

Next we provide a brief introduction to the Process Mining research field, which is followed by a review of Process Mining applications in service processes within call centers. This section concludes with a survey of the Process Mining literature dedicated to discover routing protocols in business processes.

2.3.1 Process Mining

Process mining is an emerging research discipline that bridges computational intelligence and data mining on the one hand, and process modeling and analysis on the other.

Process mining techniques can be used for process discovery and conformance checking. Process discovery [32, 33] can be used to automatically construct a process model reflecting the behavior that has been observed and recorded in the event log. Conformance checking [24, 31] can be used to compare the recorded behavior with some already existing process models to detect possible deviations. Both types of analysis may serve as input for designing and improving business processes.

Event Logs - Typically event logs contain information about the start and completion of process events together with related context data (e.g. actors and resources). Event context data is essential to discover factors that affect activity realizations in the process. Van der Aalst et al. [30] describe four levels of event contexts: the instance context, process context, social context and the external context.

Let us illustrate these context levels in a service process provided by a call center to serve class i calls. The instance of the process is the call, and the instance context data includes the call and the customer attributes. The process context data refers to the data related to the class i queue and its service resources (agent pools qualified to serve class i calls). The process context data may include the queue length and the number of available agents. The social context considers all the process external factors directly related to the service process, such as the queue

state of the other classes that may compete for common resources. Finally, the external context captures factors that are part of an ecosystem that extends beyond the call center sphere, such as weather or seasonal effects.

Process Mining techniques tend to look at process instances in isolation, and only a few studies in the area exploit context data. For examples of context-data applications we refer the reader to Werf et al. [35] and Folino et al. [6]. In [35] a context-aware approach is developed to check compliance in a publication review process; Folino, Guarascio, and Massimo [6] describe a general predictive-clustering computation scheme, meant to detect different context-related execution scenarios.

2.3.2 Process Mining of Service Processes in a Call Center

It is natural to see service to customer calls as business processes, where it is required to serve a customer call a set of coordinated activities. For example, a customer enters the call center through the IVR (Interactive Voice Response); then the call is transferred to the queue and subsequently to the agent; it could go back to the IVR, or to another queue or agent, etc.

Process mining techniques are valuable in discovering service processes. In the last several years the SEE[3] Laboratory at the Technion has conducted research projects that adopted Process Mining techniques to call center operations [5]. As part of these projects a semi-automatic process discovery package (SEEgraph) was developed, enabling users to produce real time flow graphs and animations.

Figures 2.2, 2.4 and 2.3 are snapshots of animations of the flow in a call center service process[4]. These figures illustrate some different perspectives of analysis of such processes.

Figure 2.2 represents the customer process perspective—the calls' flow. This figure depicts the paths that customers travel within the service system. In our example there are three types of call activities: *IVR service*, *Customer Queue* (wait to be served by agent) and *Customer Service* (service by an agent). The activities are represented in the graph by rectangles and are connected by edges that indicate

[3] SEE = Service Enterprise Engineering

[4] The snapshots are taken from SEEgraph animations which are available in Youtube **SEE Lab - Technion** channel https://www.youtube.com/channel/UCrIlkmNOOGwbNZ1lsUqh8Qw

where a customer may flow. The circles in the edges represent customers in the system at the snapshot time.

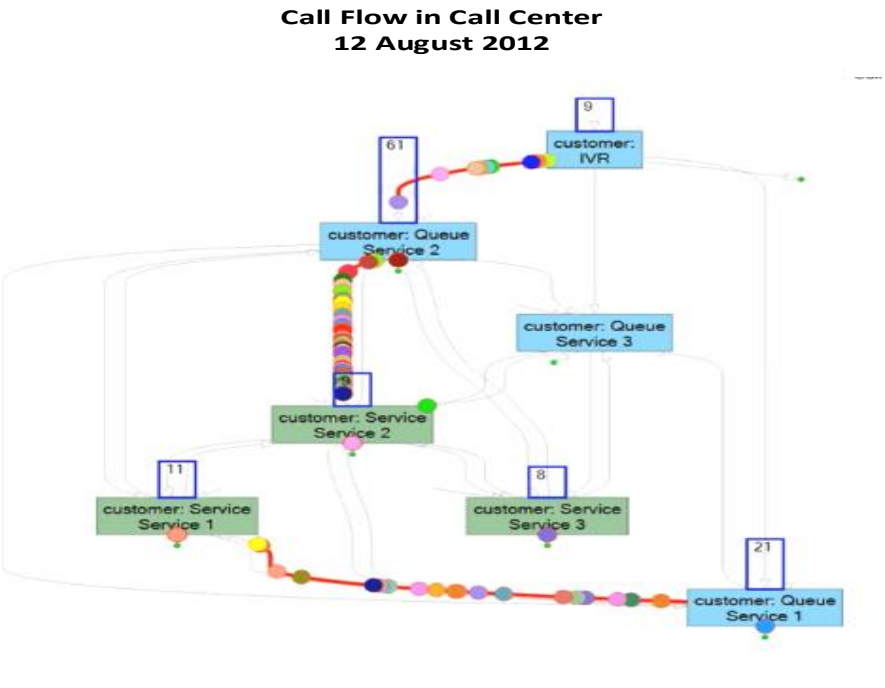

Figure 2.2: Customer Process - a snapshot of the customer flow in a call center of an Israeli bank, 12 August 2012. Available at https://www.youtube.com/watch?v=-ik5kA7aLGg

Dually to the customer flow, Figure 2.3 depicts the agent flow. The agent activities in this example are: *Business Line* (service to customer), *Agent Ready* (waiting for customers), *Paperwork*, *Idle*, *Break*, *Non-Business Line* (phone call by an agent, who does not serve a customer).

Figure 2.4 is a Process Mining representation of the SBR design and the routings in a call center. This graph represents the queues and the agent pools; the edges represent the pool skills and the call flows (routings) from the queues to the pools.

The flow diagrams, like the ones in the previous examples, are useful to analyze the design of the process (activities and edges), and the control realization (agent and call flows). In this SBR-Mining work, the flow realizations data are used to

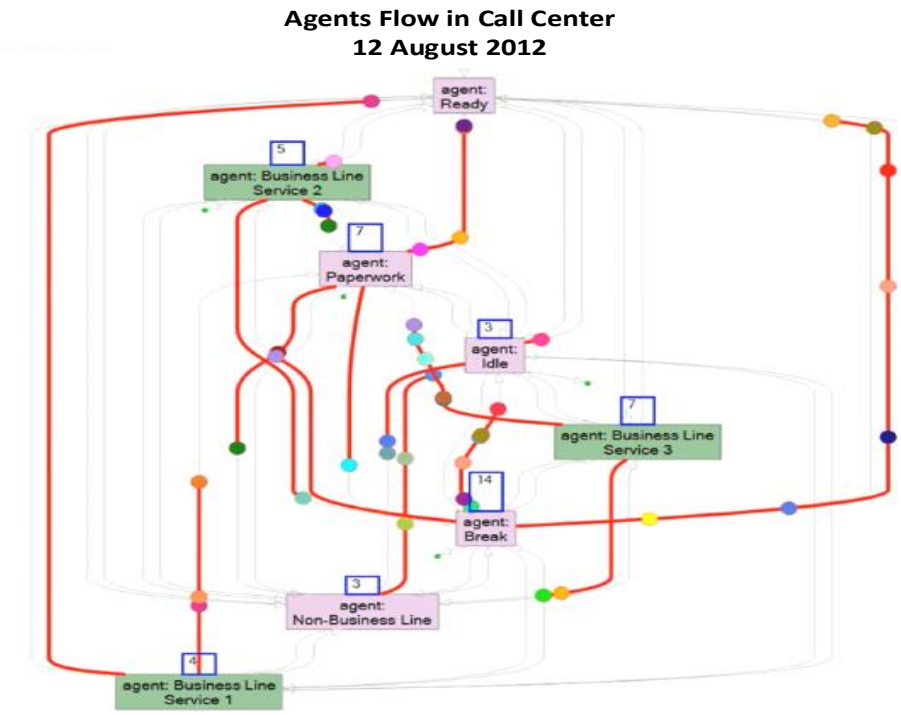

Figure 2.3: Agents Process - a snapshot of the agent flow in a call center of an Israeli bank, 12 August 2012. Available at https://www.youtube.com/watch?v=-ik5kA7aLGg

learn how the flow varies under different system conditions.

Recent works applied Process Mining techniques to study call center service processes. Senderovich et al. compared in [28] Process Mining and queueing-theoretic techniques to predict delays in service processes. Here the authors demonstrate how queueing theory gives rise to delay predictors that can be used to improve the common Process Mining predictors based on system state regression. Another work in the same spirit compared different approaches to predict agent schedules to calls [27]. This work studied the performance of queueing theory scheduling protocols and machine learning algorithms to predict agent schedules.

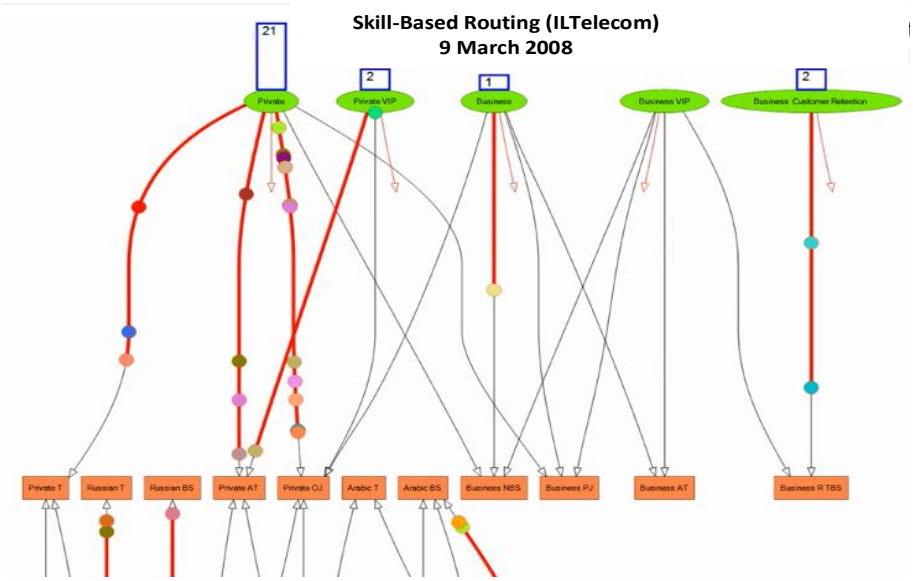

Figure 2.4: Skill-based Routings - a snapshot of the routing of calls to agents in a call center of an Israeli telecom, 9 March 2008. Taken from video at https://www.youtube.com/watch?v=1A6-jzS_scI, minute 1:41.

2.3.3 Discovering Control Protocols from Data Logs

While several Process Mining algorithms were developed for discovering instances' flow of business processes, less attention has been paid for discovering control protocols. Such protocols specify the conditions for each task in the process to be executed, including ordering tasks and assignment of resources.

Process Mining refers to protocol discovery as "decisions mining". Existing studies of this problem aim to discover routing conditions in branching points, where an instance can be routed to different activities.

A Process Mining tool to mine decisions in branching points can be found in ProM, an extensible framework that supports a wide variety of Process Mining techniques. This tool, developed by Rozinat and van der Aalst [25], defines each branching point as a classification problem, whose classes are the point branches, and the prediction features are instance attribute variables defined by the user. To build a classification model, ProM applies a decision tree algorithm.

In ProM the user defines the features used in the classification learning. To extend the set of features evaluated in the decision mining problem. Leoni et al. [4]

use a specification mining tool (Daikon) to automatically create and evaluate features that are equalities or inequalities involving arithmetic expressions on multiple variables.

The ProM decision miner tool can be applied to learn call routing protocols, but it is limited to process only instance context data. When tasks (calls) are routed to limited resources (agent pools), the process and social context play an essential role in the process routing decisions, rendering the ProM tool inappropriate.

Process and social context data is used by Senderovich et al. [27] to mine agent schedules to calls. Here various methods are compared to predict agent allocations to queues of customers, i.e., how to select a certain customer class. The problem is formulated as a classification problem as in ProM, and different predictor performances are compared: (1) queueing-theory based predictors (i.e., longest queue first and most delayed customer first) and (2) machine learning based predictors (LDA, MLR, Decision trees, and Random forest). This work shows that in the studied case the decision tree based learning achieves the best results.

The work in [27] focuses on learning protocols for the selection of a customer queue to be served first, but it does not learn the rules to select the waiting customer within the selected queue that will be served. We are not aware of any decision mining work that deals with learning selection among customers of the same class.

Chapter 3

SBR-Mining - A Method For Mining Routing Protocols

In the following chapter we first introduce a model to represent the SBR problem, as described in Section 2.2. We then develop a method for learning the routing rules from event logs of service agents and customer calls.

The proposed method is presented under the usual notation for scheduling networks, enabling a general view of the problem and application of the method in areas other than call centers, for example in data processing by web servers or scheduling patients in hospital emergency rooms.

Section 3.1 introduces a general network model for the problem. Section 3.2 describes the steps of our proposed method for learning routing protocols in the queueing network.

3.1 Model Description

3.1.1 SBR System Design in SBR-Mining

The SBR design of a system is defined as the partition of customers into classes and agents into pools. In the literature, it is usually assumed that calls and agents, in their corresponding pools, are independent and statistically identical. Under this assumption, the SBR literature assumes that calls and agents, within a class or a pool, are routed via FIFO; research then focuses on designing protocols to prioritize between system classes and pools, namely, to select which queue will be

served next, or which agent pool will serve the next call.

In reality, the situation of having FIFO classes and pools does not exist. Such protocols would prioritize customers by their waiting-time and agents by their idle times. Instead, it is common to find protocols that consider customer attributes, such as their waiting time for previous service interactions, predefined priority classifications, or measures related to the expected revenue from each customer.

We shall refer to the latter (unrealistic) design as "iid-within-pools-and-classes". Such designs would be typically difficult to fit to real-systems.

In our data-based approach, we shall start with a specific (real) SBR-Design. In such a specification, customers within a class are potentially served by the same group of agent-pools; and agents within a pool potentially serve the same group of classes. (Note that in such a specification, there are no iid assumptions within a class or a pool.) To recapitulate, our SBR-Mining approach is based on a given design of the real system studied. From the given design, the model identifies which groups of calls and agents share the same routing paths. Thus, SBR-Mining does not adopt the idd assumption.

In SBR-Mining, we learn the protocols to prioritize calls and agents within their classes and pools. If it is recognized that a partition is composed of two or more groups of customers (or agents) with different properties and different priorities, we can redefine the system partition separating the original class (or pool). For example, if we find that a given class includes VIP and Regular customers, and VIP customers are always served before Regular customers, we can redefine the design separating the customers into two new classes.

3.1.2 Inter-Queue Topology as a Network

Network notation is commonly used to describe call center processes. Senderovich's thesis [27] describes various call center processes using networks. Then the SEEGraph package provides a variety of network schemes to semi-automatically generate diagrams from event logs, emphasizing distinct aspects of system operations (see the SEEGraph examples in 2.3.2). The network notation proposed here focuses on the study of routing protocols.

In our SBR network, agents and calls are treated as similar items. We argue that the flow paths in the system of agents and calls are abstractly identical for the

SBR routing problem—both of them follow similar three main states: waiting to be routed, in service, and other states (see Figures 2.2 and 2.3). Furthermore, both agents and calls may leave the waiting state for beginning a service but also for flowing to other states. From now on the term $item$ will be used as a general term for a call or an agent waiting to be routed.

In a similar manner to the SBR flow representation in Figure 2.4, queues and agent pools are defined as vertices, consisting of waiting calls or available agents respectively. The reason to look at waiting calls and available agents is that there are items that can be routed by the protocol. Note that an agent vertex here consists of only the available agents in a class, which differs from the *agent pool* term that refers to all the agents in a class, including available and unavailable ones.

The equivalence between calls and agent routings leads one to consider the call center queueing network as an undirected graph: the edges connect agent vertices with call vertices in the same way the arrows do in Figure 2.4, connecting call queues to pools of agents who can serve the calls.

Formally, a network is an undirected graph (V, E), where the set of vertices V consists of the union of the customer classes set $\mathcal{I}$ and the agent pools set $\mathcal{J}$ defined in Section 2.2; the set of edges is described by a symmetric matrix $E_{v_1 v_2}$, where $E_{v_1 v_2} = 1$ if routings from vertex v_1 to vertex v_2 and vice versa are possible under the system design. Otherwise $E_{v_1 v_2} = 0$. If $v_1 = v_2$ then $E_{v_1 v_2} = 0$. In addition, let V^v be the set of neighboring vertices of vertex v, namely the set where v items can be routed: $V^v = \{v' | v' \in V \wedge E_{vv'} = 1\}$.

Figure 3.1 illustrates the network representation of an N design. In this example, $V = \{1, 2, 3, 4\}$, and $E = \begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}$

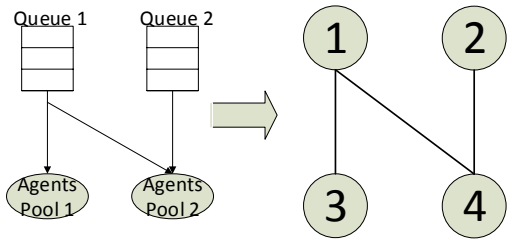

Figure 3.1: N design as a network, which consists of four vertices and three edges.

The network schema is useful to describe a model for the network routing

protocol below. However, the studied system is based on the network, but is not reduced to it. The network routing rules are learned based on logs of all the item events in the system, including events in the queueing network and events outside the queueing network (e.g. recorded messages).

3.1.3 Routing Protocols

A routing protocol evaluates at each point in time whether any, and if so which, items are to be routed. The protocol evaluations can lead to routing waiting customers or idle agents to service, or to the decision that no routing is to be executed. These evaluations answer two questions:

- Should any item be routed at time t?
- Given that an item is to be routed at time t, which items are routed?

Our model assumes that routing protocols have **No Induced Idle Time (NIIT)**. This means that there will never be a waiting item in a vertex, while a compatible item is waiting in the network. Specifically, in the call center network this means that there will never be a waiting call in queue, jointly with a skilled agent available to serve this call.

Routings in our model are also non-preemptive, as defined previously—a customer service is not interrupted to serve another customer.

NIIT and non-preemption define exactly when the protocol executes routings: whenever an item arrives and there is an available compatible item in any neighboring vertex. Therefore, the question *"Should any item be routed at time t?"* has a definite answer by the NIIT assumption. Our work focuses on the second question, *"Given that an item is to be routed at time t, which items are routed?"*. It is possible to also study routing protocols that do not require NIIT, but these protocols are beyond the scope of the current work.

In addition, we assume that the item arrivals to each vertex are independent random processes with continuous interarrival distributions. Therefore, since no more than one item arrives at each point in time t, no more than one item can be routed at time t.

A central characteristic of our SBR-Mining approach is that it does not assume that items that enter the same vertex have the same priorities; in our approach VIP

and regular customers could be in the same vertex. It is assumed that the vertices group items with the same feasible routing paths (connections) but not necessarily with the same priorities.

Upon the arrival of an item to vertex v at time t, the routing protocol processes the available data about the system state and the arrived item to define whether and how this item has to be routed. Let x_t be the system "state" at time t, a description of the information available at time t, including data on a possible arrival at this point in time, and let d be a routing decision at time t, which can be an empty set if no routings are executed, or a pair of matched items (i, j), when i is the arrival item at this time and j is a waiting item selected to enter service.

The following paragraphs define the network routing protocol in the introduced model, and describes how it is decomposed into simpler vertex protocols.

Definition of Network Routing Protocol. *A network routing protocol is a function $R : \mathbb{X} \to \mathbb{D}$ that maps system states $x \in \mathbb{X}$, to routing decision $d \in \mathbb{D}$.*

Furthermore, item routings can be divided into two routing levels:

1. **Inter-vertex Routing**: this is the selection of the vertex where the arriving item is to be routed.

2. **Intra-vertex Routing**: refers to the selection of the specific item from the vertex determined by the inter-vertex routing, which is to handle the arrival item.

Before formalizing these definitions of routing protocol, we illustrate the definitions above in the call center network. Suppose that a new call arrives to a queue. If there are available agents who can handle the call, according to the NIIT assumption the call is routed immediately (service routing). The inter-vertex routing defines which agent pool will serve the call, and the intra-vertex protocol selects an available agent from the selected pool to serve it.

Inter-vertex Protocol definition. *The vertex v inter-vertex protocol is a function $F^v : \mathbb{X}^{V^v} \to V^v$: its argument is a system state $x \in \mathbb{X}^{V^v}$ for which at least one of the vertices in V^v is not empty, and its value is a vertex v' selected by the protocol to route the arriving item. The inter-vertex protocol has the property that $F^v(x_t) \in V_t^v$, with V_t^v being the set of nonempty vertices in V_v at time t.*

In SBR-Mininig, inter-vertex protocols are described through preference score functions, which gives score values to the vertices that can be routed, the routed vertex being the one with the higher score value. Formally, a preference score function is a function $P_v : (\mathbb{X}^{V^v}, V^v) \to \mathbb{R}$, for which:

$$F_v(x_t) \in \underset{v' \in V_t^v}{\operatorname{argmax}} P_v(x_t, v').$$

Preference score functions are learned from data, and through these learned functions the inter-vertex protocol is characterized.

The preference score function here is equivalent to the index functions introduced in the previous chapter, but without the restriction of using only local data. The Intra-vertex Protocol defined below is also formulated through a performance score function.

Intra-vertex Protocol definition. *Letting X^v be the set of system states for which vertex v is not empty, and I^v the set of items that visit vertex v during the studied period*[1]*, the vertex v intra-vertex protocol is a function $f^v : \mathbb{X}^v \to I^v$, which processes the system state information $x \in \mathbb{X}^v$ to select an item from I^v to enter service. The intra-vertex protocol of vertex v has the property that $f^v(x_t) \in I_t^v$, where I_t^v is the set of items at vertex v at time t.*

Similarly to the preference score functions described for learning inter-vertex protocols, intra-vertex protocols are described by preference score functions denoted $p_v : (\mathbb{X}^v, I^v) \to \mathbb{R}$, for which:

$$f_v(x_t) \in \underset{i \in I_t^v}{\operatorname{argmax}} p_v(x_t, i).$$

Let us formulate the network routing protocol $R : \mathbb{X} \to \mathbb{D}$ as a composition of the intra- and the inter-vertex definitions. For that, let A_i be the arrival time of item i to its vertex. Then we have

$$R(x_t) = \begin{cases} (i, f_{v'}(x_t)) | v' = F_v(x_t), & \exists v \in V, i \in I^v | t = A_i \ \wedge \ |I_t^v|. \sum_{v'=1}^{|V|} E_{v,v'} > 0 \\ \emptyset, & \textit{otherwise.} \end{cases}$$

[1]The I^v notation should not be confused with the notation for idle agents in Section 2.1.

3.2 Method

In this section, we describe our method to learn the routing protocols of a call center from its event logs. Chapter 4 presents an application of the method to two call center databases.

Call centers typically use simple and intuitive SBR protocols. Therefore, our method is specially designed to find simple and intuitive routing rules. To achieve this purpose, the method details an interactive trial-and-error analysis where a human analyst formulates and evaluates routing rules and arguments. A different approach to learn routing protocols would be to define a wide set of possible routing rule arguments and building an algorithm to automatically discover the protocol and its arguments. Such an approach, however, is likely to lead to protocols that are complex, not intuitive, and hence will differ from the real protocol. Let us remark that the latter automatic discovery approach can still be considered a trial-and-error approach, where only one trial iteration is done. Our model and method, as described in this section, still have a significant value for this approach as well.

Our method assumes that there exists a routing protocol $R : \mathbb{X} \rightarrow \mathbb{D}$, and that the system state description processed by the protocol is a function of the event logs in the studied database. Network protocol learning consists of learning the intra-vertex and inter-vertex routing functions of each vertex.

An algorithm to learn vertex routing protocols is introduced below. This algorithm aims to discover the two preference score functions $p_v(x_t, i)$ and $P_v(x_t, v')$, which describes the vertex protocols. The function argument x_t was defined to be a general system state description which includes all the available data at time t. In fact, to characterize a specific vertex routing protocol, this argument must be specified as it serves as the system state variable. Let x^{p_v} and x^{P_v} be specified vector state descriptions used respectively by the $p_v(x_t, i)$ and $P_v(x_t, v')$ preference functions. Our learning algorithm aims to learn the preference score function and its corresponding system state parameter x^{p_v} and x^{P_v}.

The complexity in discovering routing rules lies in discovering the state vectors x^{p_v} and x^{P_v}. This complexity is originated by the fact that these vectors can be composed of a wide set of measures and variables, such as item attributes, queue load measures, etc. Note that also simple measures, such as the waiting time of items, can be defined in different ways. For example, the waiting time could be the

time in queue, or the time since the entry to the call center, including VRU time.

The fact that routing protocols are simple and system state vectors can be defined in several different ways, makes the task of discovering the state vector the central challenge in SBR-Mining.

3.2.1 Description of the Algorithm

The learning algorithm is essentially similar for the inter- and the intra-vertex protocols. We now introduce a general description of the algorithm, followed by specifications for each type of protocol learning.

The proposed method, that works in a trial-and-error approach, is a cyclic algorithm composed of three steps: (i) protocol formulation, (ii) protocol testing and (iii) exploration that seeks new protocol formulation:

- **Formulation**: Consists of defining a preference score function for the studied vertex (p_v or P_v) and defining its concrete system state description vector (x^{p_v} or x^{P_v} respectively).

- **Protocol Test**: Tests the preference score function goodness-of-fit to data-logs. For that, the system logs are processed to find routing decisions executed by the studied protocol and to compute how well those routings fit the formulated score function.

- **Exploration**: Aims at learning how the formulated protocol can be improved. It tests the capacity of new state variables to explain the difference between the routings observed in logs and the routings defined by the formulated protocol. Variables that explain differences in routings are included in the new formulated protocol.

  To test system state variables, descriptive models of the routing fit are developed, where the explaining variables are the system state variables to be tested. Then, these models are used to analyze how the variables explain the routing fit and to select one of the tested features.

3.2.2 Learning of Intra-vertex Protocol

Intra-vertex protocols refer to the decision of which item in the studied vertex will enter into service next. The learning method for this kind of protocol is detailed

below.

Intra-vertex protocols could be classified into two categories: (i) homogeneous item protocols, distinguishing items only by the arrival-to-vertex time stamps, e.g. FIFO (*First In, First Out*) or LIFO (*Last In, First Out*), and (ii) heterogeneous item protocols, where the selection of items is based on vertex external data, such as item expected revenue, time spent by the item in previous vertices, or item class.

Common homogeneous item protocols are easy to be recognized recognize via data analysis. In homogeneous vertices, the natural protocol to be used is FIFO. Heterogeneous item protocols are harder to detect from data since these may be based on item value measures that are not included in operational databases, or based on complex computations of the items' history in the system.

In some cases, heterogeneous protocols differentiate between items by some categorical attribute, such as item quality group (regular, VIP,..) or previous service type. In these cases we propose to split the heterogeneous vertex into one-category vertices, which group homogeneous items.

Formulation

The intra-vertex protocol function f_v is described through a preference score function p_v, for which

$$f_v(x_t^{p_v}) \in \underset{i \in I_t^v}{\operatorname{argmax}}\, p_v(x_t^{p_v}, i).$$

The learning method introduced below studies the intra-vertex protocol through its preference score function. Note that the general x_t symbol used in the original protocol definition is here replaced by the specified system state vector $x_t^{p_v}$, which reduces the general system state description to a vector that contains only the system state variables that define the intra-vertex routings.

In the formulation step, a preference score function p_v and its $x_t^{p_v}$ argument are formulated to be tested and to be improved after an exploration cycle.

Testing

The purpose of the testing step is to quantify how well the routing protocol fits the routings derived from the event logs. The goodness-of-fit measure enables comparison between formulated protocols and a general quantification of the learning process results.

To define how well a routing protocol fits the data, we use the **skips** and **slips** concepts, defined in [12]: *"if item i enters the vertex before item j enters the vertex, but item j leaves the vertex before item i, then item j **skips** item i and item i experiences a **slip** from item j"*.

The skips and slips in [12] are defined as violations of the FIFO protocol. Hence, many skips and slips suggest that the FIFO protocol does not fit the routing derived from the system logs. Below we generalize these concepts that are applicable not only for testing the FIFO protocol, but also any intra-vertex protocol.

Skips and Slips definition. *Suppose that at time t items i and j wait in vertex v and one of these items is routed; suppose also that item i has a lower preference value than item j. Denote by r_i (r_j) the time when item i (j) enters service. Then $skips_{ij}^t$ indicates whether item i is routed before j (i skips j) or formally:*

$$skips_{ij}^t = \mathbb{1}_{\{r_i=t\}}, \quad \forall\{i,j \in I_t^v \ \wedge \ p_v(x_t^{p_v}, j) > p_v(x_t^{p_v}, i) \ \wedge \ t = min(r_i, r_j)\}.$$

Accordingly, $slips_{ij}^t$ is defined by

$$slips_{ij}^t = skips_{ji}^t.$$

Due to the symmetry between skip and slip indicators, the following steps will be described through the skip indicators, but slips could be used as well.

Skip indicators are defined for **skip opportunities**, namely, situations when two items wait in queue at the same time and one of the items is routed before the other. The indicator values indicate whether the first item selected from the queue is not the item with the higher priority defined by the tested protocol.

Skip opportunities are observations of the system routing protocol; the set of skip opportunities form the SBR-Mining learning sample. Note that when an item is selected to leave the queue, a skip opportunity observation is registered for each waiting item. Therefore, routings executed when the vertex is highly loaded are represented by more observations in the sample than routings executed when the vertex is lightly loaded. This sampling approach increases the weight of routings executed during heavily loaded periods, focusing the learning process on these routings.

To illustrate the concepts introduced above, consider a case where we are testing the FIFO protocol in a queue with 100 waiting items and no new arrivals. For

this example, the items in queue are denoted by index $i = 1, 2, 3, ..., 100$ according to their arrival order ($i = 1$ is the first item to join the queue). In this example, the last item to join the queue, item 100, is routed first, and afterwards all the items are routed under a FIFO protocol.

When item 100 is routed, 99 skip opportunities are generated, and their skip value is 1, indicating that skips actually occur: $skips_{100,i}^{r_{100}} = 1, \forall 1 \leq i \leq 99$. The second item to be routed is item 2, generating 98 skip opportunities, whose skip value is 0: $skips_{i,1}^{r_1} = 0, \forall 2 \leq i \leq 99$. In the same way, the routing of item 2 generates the indicators $skips_{i,2}^{r_2} = 0, \forall 3 \leq i \leq 99$; and so on, until item 99 is routed last.

Goodness-of-Fit based on Skips and Slips - In the sample of skip opportunities, each skip value indicates if the evaluated protocol fits the corresponding routing observation.

The skip values are aggregated to compute general goodness-of-fit measures. There are various ways to do that, from which we chose two goodness-of-fit measures.

The first measure is the **skipping items rate**, which measures the number of items that skip relative to those that had skip opportunities:

$$\frac{|\{i : i \in I^v \wedge \sum_{j,t} skips_{ij}^t > 0\}|}{|I^v|}.$$

The second measure proposed is the **skip rate**—the rate of skip opportunities where a skip occurs:

$$\frac{\sum_{i,j,t|i>j} skips_{ij}^t}{|\{(i,j)|\exists skips_{ij}^t\}|}.$$

Let us demonstrate these measures using our previous example. In that example, we get a sample of 4950 skip opportunities, from which only 99 skips occur, so the skip rate is $99/4950 = 2\%$. The skipping item rate is $1/99 = 1.01\%$, since from the 99 items that had the opportunity to skip, only one skipped.

Exploration

The exploration step is the part of the trial-and-error method where errors are analyzed to formulate an improved protocol. In this part, the analyst studies the errors

of the protocol evaluated in the last iteration, namely the observations of skip opportunities for which the skip indicator value is 1. The exploration consists of finding explanatory variables for skip values. The idea is that system state measures correlated with the skip values can be used to formulate a new improved protocol.

To analyze the protocol errors, the analyst builds descriptive models of skip values, where system state variables describe the skip values in the sample. The purpose of the descriptive model is to recognize relations between system state variables and the errors in the protocol. Using these relations, the analyst can formulate a new protocol to be tested in a forward iteration.

To build descriptive models of the skip values, different machine learning algorithms can be used. We recommend the use of algorithms that lead to simple and comprehensive models, since our method aims to find simple routing protocols. Decision tree algorithms, such as Cart [19, Chapter 9], can be appropriate for this purpose. In addition, in [27] it was shown that decision trees are good in identifying routing protocols.

When the model recognizes categorical attributes of items as good explaining variables of skips, the vertex can be split into new vertices according to the recognized attribute. This simplifies protocol analysis, and creates a more detailed description of the different groups of items in the system.

The learned relation between state variables and the protocol errors are implemented splitting the vertex items, if necessary, and formulating a new preference score function. The need for a new learning cycle is determined by two criteria—the goodness-of-fit reached and the discovery of new insights in the exploration phase.

To conclude the learning description of the intra-vertex preferences and topology, Figure 3.2 illustrates how an N network is further decomposed after finding that Vertex 2 in fact groups two different item classes. These 2 groups could be identified after finding that items who skip share a characteristic that other items do not, for example, a customer VIP or regular classification. In the N network case in Figure 3.2, Vertex 2 in the original design is split into two new vertices, Vertex $2a$ and Vertex $2b$, and the intra-vertex protocols are studied over a new network consisting of 5 vertices.

Algorithm 1 Intra-vertex Preferences Learning and Design Evaluation

1: Define an intra-vertex score function p_v and its system state vector $x_t^{p_v}$ (as in Section 3.2.2, Formulation Part).

2: Compute skip variables for the sample of skip opportunities. Evaluate goodness-of-fit measures (as defined in Section 3.2.2, Testing Part).

3: Build descriptive models of skip indicators and analyze skip explanatory variables (as described in Section 3.2.2, Exploration Part).

Evaluate if the vertex can be forward decomposed into different item classes and can be split into new vertices. In this case return to Step 1 for each one of the new vertices.

4: Based on the insights from the analysis, define a new intra-vertex score function p_v and vector $x_t^{p_v}$ (as in Section 3.2.2, Formulation Part) and return to Step 2. If no insights are reached, stop the learning process.

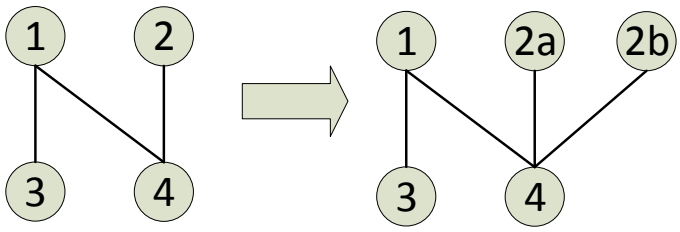

Figure 3.2: Example of a vertex split in an N-design network. Here vertex 2 is split into vertices $2a$ and $2b$.

3.2.3 Inter-vertex Protocol Learning

The previous section introduced an algorithm to evaluate the network topology and to learn intra-vertex protocols. This section describes an algorithm for learning inter-vertex protocols, that select the vertices to which items are routed.

Figure 3.3 illustrates how the network from Figure 3.2 is decomposed into five inter-vertex functions: one gets three I-design networks (the vertex is connected to only one vertex) and two V-designs (the vertex is connected to 2 or more vertices). In I-networks the routing is trivial while in V networks the routing is a classification problem.

Denote by v the vertex whose inter-vertex protocol is studied. Recall that V_v

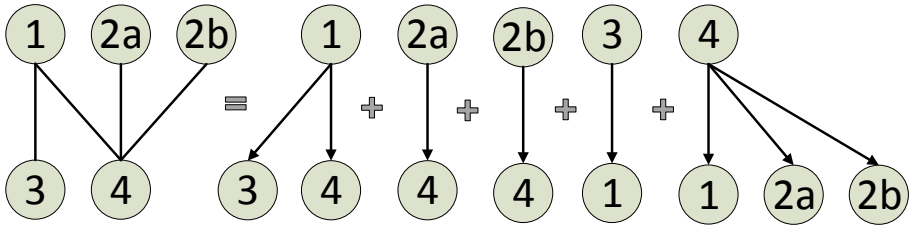

Figure 3.3: Network decomposition to inter-vertex routing protocols. The network in Figure 3.2 is decomposed into five different networks, each one with its own inter-vertex routing protocol.

was defined in Section 3.1.2 to be the set of vertices neighboring v, namely the vertices that can be selected by the v inter-vertex protocol. Recall also from Section 3.1.3 that the v inter-vertex protocol was defined as the selection of vertices in V_v, where items from v are routed.

From an abstract point of view, the inter-vertex protocols are similar to the intra-vertex ones. We can look at vertex v as a *meta-vertex* which groups *meta-items* (v neighbors) that wait to be routed. Under this view, the inter-vertex protocol, as the intra-vertex one, routes one item selected from a group of waiting items that are in the vertex at the routing time.

In general the algorithm to learn inter-vertex protocols is similar to the algorithm described to learn intra-vertex protocols. Nevertheless, there exist differences between both types of protocols that must be considered in the learning process. The comparison below is somewhat general and may not apply to every studied protocol, but it is useful in order to introduce the differences between the algorithm approaches. Later on we mention how protocols with different characteristics can be handled.

The first difference between intra- and inter-vertex protocols is that, while the intra-vertex protocol is expected to route homogeneous items, the inter-vertex protocol aims to distinguish between the vertices in V_v and to prioritize some classes of waiting items.

The second difference is in the number of different items routed by each type of protocol. Informally, we can say that an inter-vertex protocol always routes the same group of a few meta-items (vertices), while intra-vertex protocols commonly

route several different items (tens to a few hundred agents in agent vertices or several thousands of calls per day in call vertices).

These two differences do not affect the general structure of the trial-and-error algorithm, but do affect the algorithm approaches: while, in the intra-vertex algorithm, we learn preference functions of **general** items i; in the intra-vertex protocol we learn preference functions of **specific** meta-items, the vertices in V_v[2]. Below we detail the learning algorithm for the inter-vertex protocols.

Formulation

The first step of each trial-and-error cycle is formulating the routing protocol $F_v()$ and its parameters. The inter-vertex function was described by a preference score function P_v and a system state vector $x_t^{P_v}$:

$$F_v(x_t^{P_v}) \in \underset{v' \in V_v^t}{\operatorname{argmax}} P_v(x_t^{P_v}, v').$$

The protocol tested in the trial-and-error method is defined through its preference score function. In this step the preferences function P_v and its system state parameter x^{P_v} are formulated.

Since we expect that an inter-vertex protocol will give different priorities to the vertices in V_v, the formulated preference score function is likely to include indicators for each vertex that can be routed. These indicators are multiplied by expressions to increase or decease the score values of each vertex in a different way. For example, consider a preference score function for vertex v, for which $V_v = \{1, 2\}$, and its routing protocol routes vertex 2 only; in vertex 2, there are always more than 6 items. Here we define indicators for vertices 1 and 2, $\mathbb{1}_{\{v'=1\}}$ and $\mathbb{1}_{\{v'=2\}}$, and formulate the preference score function to be: $P_v(x_t^{P_v}, v') = \mathbb{1}_{\{v'=1\}} + 2.\mathbb{1}_{\{v'=2\}}.\mathbb{1}_{\{|I_t^2|>6\}}$.

Test

In the test part, the goodness-of-fit of the formulated protocol is evaluated. Due to the preference score computation may be different for each vertex in V_v; the

[2]Our inter-vertex algorithm aims to learn routing protocols that route the same few items all the time. Learning inter-vertex protocols of vertices with several neighbors could be convenient applying the intra-vertex algorithm, and in the same way, intra-vertex protocols that deal with a few returning items, like small agent pool vertices, could be learned using the inter-vertex algorithm.

testing part aims to provide goodness-of-fit measures for routing observations of each vertex. Such measures may indicate which vertices' score computations could be improved in forward iterations.

For inter-vertex learning *vertex skips and slips* concepts are defined. In contrast to the skips and slips indicators defined for learning intra-vertex protocols, which are defined for general items $i, j \in I^v$, the vertex skips and slips defined here are defined for specific vertices in V_v.

Skips and Slips definition. *Suppose that at time t vertices v_i and v_j are in V_v^t, and an item from one of the vertices is routed; suppose also that vertex v_i has a lower preference score $P_v()$ than vertex v_j. Denote by $r_{vv_i}^t \in \{0,1\}$ a variable that indicates if at time t an item was routed from the studied vertex v to vertex v_i. We define $skips_{v_ij}^t$ to be an indicator which indicates if vertex v_i skipped v_j at time t, or formally:*

$$skips_{v_iv_j}^t = \mathbb{1}_{\{r_{vv_i}^t=1\}}, \ \ \forall t, \ \forall v_i, v_j \in V_v^t | P_v(x_t^{P_v}, v_j) > P_v(x_t^{P_v}, v_i) \wedge \max(r_{vv_i}^t, r_{vv_j}^t) = 1.$$

Accordingly, $slips_{v_iv_j}^t$ is defined by

$$slips_{v_iv_j}^t = skips_{v_jv_i}^t.$$

Skips and Slips based Goodness-of-Fit - The goodness-of-fit of the inter-vertex protocol is expressed by aggregation of the skip values defined above. Several measures could be defined to be these goodness-of-fit vector values. Here we use the skip rate measure defined in the intra-vertex routing learning. The second measure used in the intra-vertex protocol, the skipper rate, is not relevant for protocols that route only a few vertices.

We propose to measure the fitting separately for observations of each routed vertex, then the analyst can focus the exploration on the vertices with the worst fitting. For example, the goodness-of-fit can be expressed by a vector of size $|V_v|$ composed of aggregations of the skip values of each vertex in V_v. When $|V_v| > 2$, it may be informative to compute also the goodness-of-fit of each pair of vertices.

Exploration

The exploration step to improve the formulated preference score function is similar to the exploration described in the intra-vertex protocols learning algorithm. It

builds descriptive models to relate new system state variables to the protocol error indicators, namely the skip values.

In the inter-vertex protocols learning, the goodness-of-fit vector provides information about the vertices for which the preference function shows a worst fitting. These are the vertices where the exploration should focus on ??. Then, the sample of skip opportunity observations processed to build the descriptive model is composed by only the observations related to the explored vertices.

In an exploration step various skip opportunity samples and explaining variables can be analyzed through different models. These models are analyzed to recognize which variables explain better the protocol errors and the insights from the model analysis; then a new routing protocol is formulated.

The algorithm ends in one of two cases—when the formulated protocol fitting to routing logs is satisfactory or when exploration models do not lead to insights that can be used to improve the formulated protocol.

Algorithm 2 Inter-Vertex Protocol Learning

1: Define an inter-vertex score function P_v and its parameters $x_t^{P_v}$. (as in Section 3.2.3, Formulation Part).
2: Compute skip variables for the sample of skip opportunities. Evaluate goodness-of-fit measures (as defined in Section 3.2.3, Testing Part).
3: Build descriptive models of skip indicators and analyze skip explanatory variables.

   The analysis may be done separately for different subsets of skip opportunity observations (as described in Section 3.2.3, Exploration Part).
4: Based on the analysis insights, define a new inter-vertex score function P_v and vector $x_t^{P_v}$ (as in Section 3.2.3, Formulation Part) and return to Step 2. If no insights are reached, stop the learning process.

Chapter 4

Application - SBR Mining in Real Data

In this chapter, we apply our SBR-Mining method in three case studies. The chapter begins by describing the studied systems and their databases. It is followed by a short discussion of practical issues in call log processing that aims at learning routing protocols. At the end of the chapter, three examples of protocol learning are described, using the SBR-Mining method.

4.1 Data Description

For our analysis, we use databases from two large call centers. The first belongs to an American bank (**USBank**), with approximately 1000 service agents at peak hours. The second one is from an Israeli telecom company (**ILTelecom**), with around 400 agents at peak hours.

The databases originated from the ongoing Data MOCCA[1] research project, conducted at the SEE[2] Center at the Technion. (For more information on the SEE Center and the DataMOCCA project see [5]).

Call center data is stored by the Automatic Call Distributors software (ACDs), which stores raw data containing the history of calls. Tables 4.1 and 4.2 describe the number of incoming calls recorded in our studied databases.

[1] Data MOCCA = Data MOdels for Call Center Analysis

[2] SEE = Service Enterprise Engineering

USBank	Total	Total Weekdays	Average per Weekday
Total # of arriving calls	218,047,488	181,032,004	271,006
# calls requesting agent service	41,646,142	37,036,994	55,445

Table 4.1: Call summary for USBank, March 2001–October 2003

ILTel	Total	Total Weekdays	Average per Weekday
Total # of arriving calls	16,583,207	12,971,371	205,040
# calls requesting agent service	2,999,191	2,740,649	43,438

Table 4.2: Call summary for ILTelecom, January–March 2008

The flow of calls and their SBR design is detailed below, followed by a description of the call logs used to learn the routing protocols.

Call Flow - The customer-originated call enters the Call Center system at a particular node, usually via a VRU—Voice Response Unit. In some applications, the call may also enter via an Information Announcement, via the Call Center voice messaging system or even directly to an agent service group. As described in Tables 4.1 and 4.2, most of the calls are in fact self-service transactions conducted or information received at the VRU, Announcement or Message stages.

At the next stage, for the customers who desire to speak to an agent, the call is transferred to be served by an agent who is capable of performing the desired service (has the required skills). The customer then either waits until an appropriately skilled agent becomes free, or else the customer abandons the call center. At completion of service by the agent, the call either ends, or has a continuation. In the latter case, in the database, the original call is divided into the first customer subcall which ends when the first service was completed, plus the remainder of the call, which may be divided into further subcalls. Figure 4.1 illustrates the calls flow in a USBank during an average weekday.

System Design - The system design is the partitioning of customers into classes and the servers into pools; each pool has a predefined set of skills. Our method to learn routing algorithms uses the system design to learn separately the routing protocols of each pool and class.

The ILTelecom system design is depicted in Figure 4.2. The design repre-

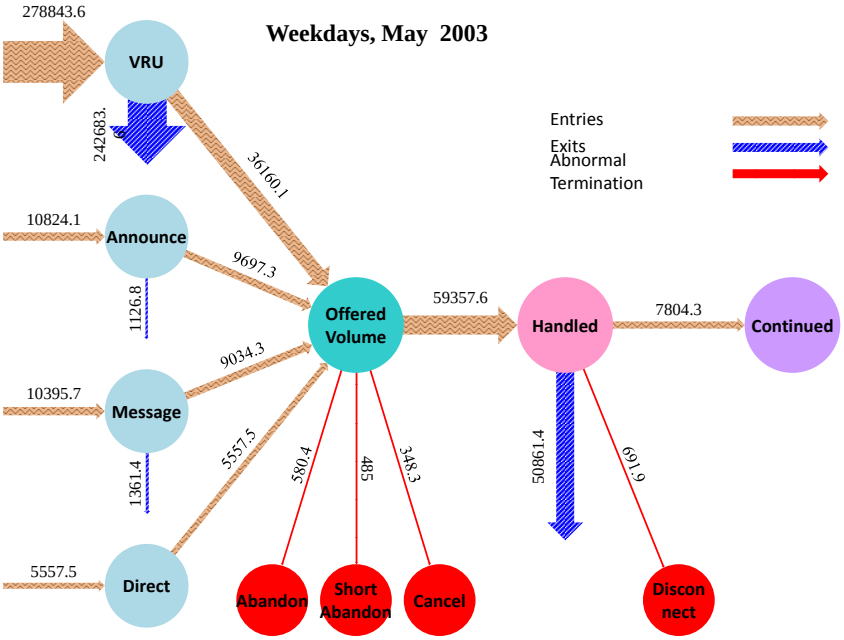

Figure 4.1: Flow Diagram of incoming calls, in a USBank during the weekdays of May 2008 (mean values).

sentation includes the customer classes (green ellipses), the agent pools (orange rectangles) and the number of calls from each class served by each agent pool (arrows connecting queues and pools). This graph is automatically produced from the call logs by the SEEGraph package.

The ILTelecom call center consists of 19 customer classes (queues) and 16 agent pools. Some customer classes, such as the various Private and Business classes, can be served by various agent pools. While other customer classes, like Technical and Financial classes, are served by a unique dedicated pool of agents. Similarly, some agent pools serve various customer classes and others serve a unique class.

The USBank SBR-design is illustrated in Figure 4.3. In this case the original database does not contain the partition of agents into pools. The design is hence taken from [20], which created it via an agent clustering algorithm. The USBank design has 9 customer classes and 11 agent pools. As can be viewed in the graph, most of the incoming calls are from the Retail class. Six agent pools serve Retail customers, some of them exclusively dedicated to serve them; and some others are

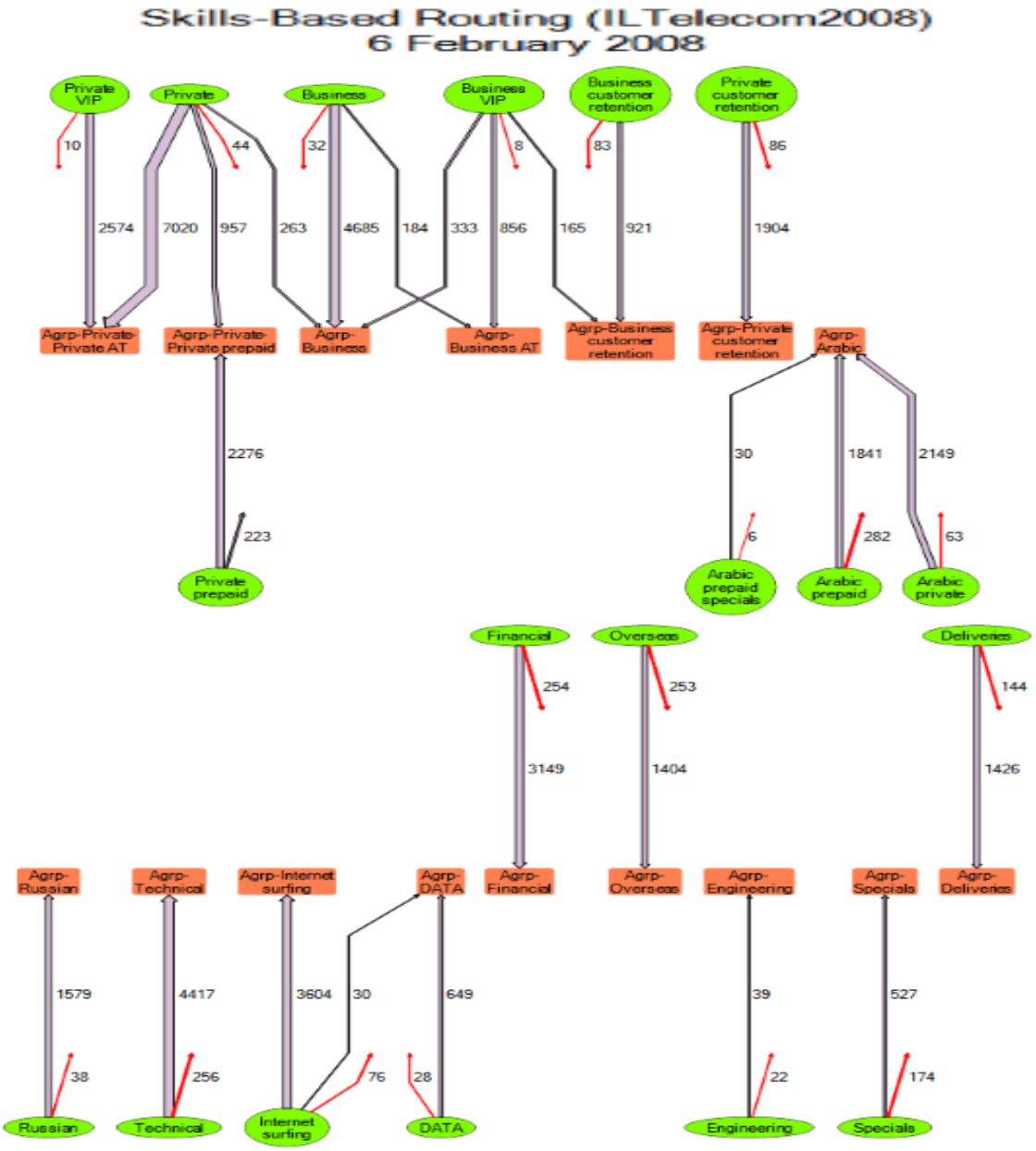

Figure 4.2: ILTelecom call center design - based on data from the 6 of February 2008. Graph produced by SEEGraph.

dedicated to serve other classes and serve the Retail class as well. Five pools do not serve Retail customers and are dedicated to other customer classes.

Data Logs - The studied databases include logs for agent and customer activ-

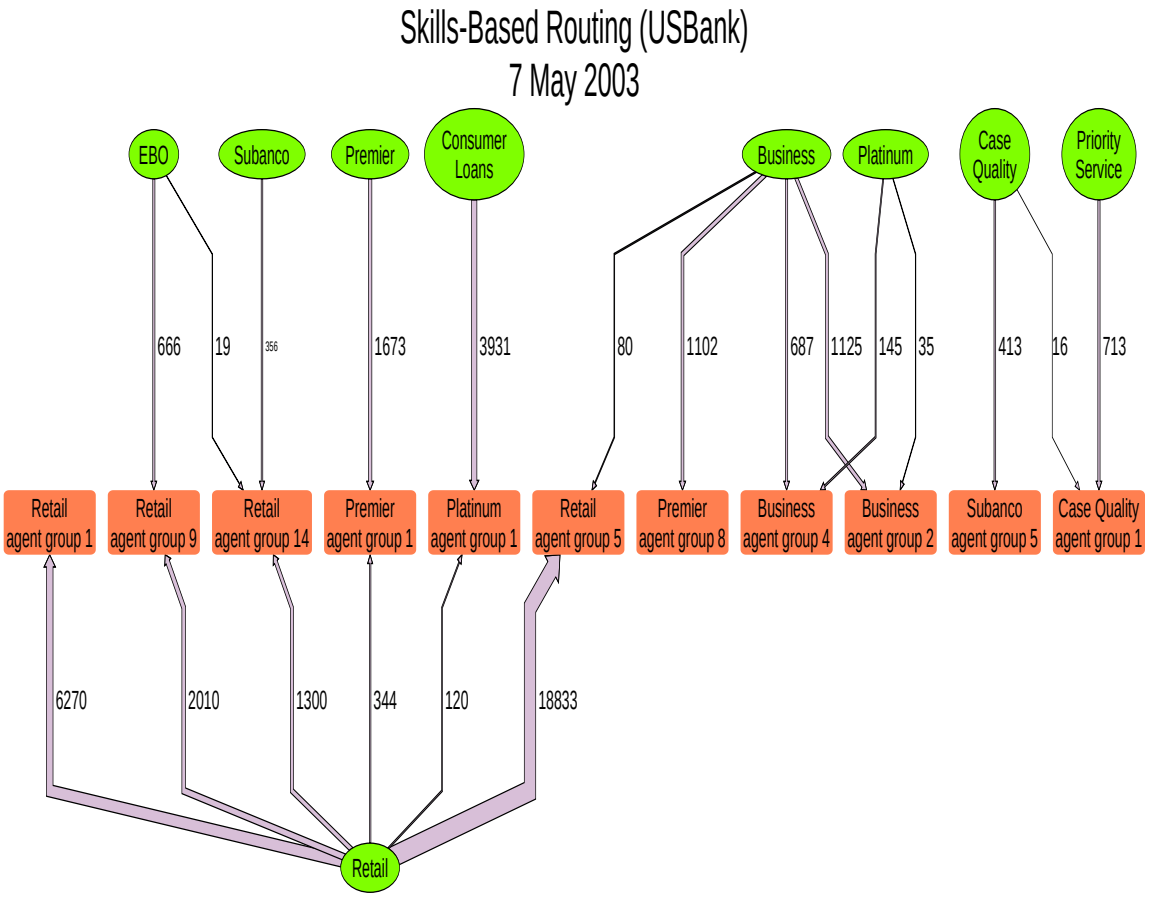

Figure 4.3: USBank call center design - based on data from the 7 of May 2007. Graph produced by SEEGraph.

ities in the system. Each log record describes an activity and includes information about the agent or the customer involved in the activity.

For example, for a standard customer call, a log is stored for the customer interaction with the VRU, and another log for the customer interaction with the agent. If after receiving service by an agent, the customer is routed to receive another service, this will be stored by additional logs. Customer logs include the following fields:

- Identifiers - call ID, customer ID, subcall number (enumerates the service interactions in the same call).
- Service class.

- Answer party type - e.g. agent, VRU, information announcement, etc.
- Answer party ID - e.g. VRU or agent ID.
- Customer node origin - physical or logical system component that processes the activity. The USBank system consists of different nodes that process calls from different geographical sites.
- Timestamps of the entry to queue, entry into service and end of service.
- Segment component durations - talk time, queue time, hold time, wrap-up time, etc.
- Exit reason - end of service, abandon, transfer or error.

Agent event logs describe agent activities, such as customer service and breaks. Below is a list of the main fields stored in agent logs:

- Identifiers: Agent ID.
- Agent pool.
- Activity - shift beginning/end, customer service, break, etc.
- Agent node origin: refers to the system component assigned to process the agent activities.
- Timestamps of the event, specifically beginning and end.
- Served party ID - customer ID, call ID, call log number.

In fact, the original logs' database includes a significantly more detailed description of the events. The logs' description here presents the main structure of the logs and the main field processed in our examples.

4.2 Data Challenges in SBR-Mining

In SBR-mining, some practical issues must be considered regarding the routings executed in the system and their log storage process. This section discusses two practical aspects that affect the derivation of meaningful results: The first one is the existence of one unique deterministic protocol during the studied period. The second practical issue discussed is quality of the data logs.

Routing Protocols—Existence and Stability

The described algorithm assumes that during the defined study period, all the routings are determined by one unique deterministic set of rules. SBR-Mining aims to discover this rule set.

In general, routing protocols defined in ACD systems are a stable setting of call center operations. The definition of new protocols imply deep analysis, research and ACD system programming, and therefore these are not changed frequently. Nevertheless, protocol definitions usually include some constants that may be recalibrated once in a while, like thresholds or the queue distribution in a QIR rule.

These protocols' parameters affect the items' preference scores, and therefore the preferences of items' classes. Since changing these parameters is simple, they are commonly tuned by a trial-and-error approach. When these changes in the routing protocols are not stored in the database, the SBR-Mining results can be strongly affected.

To reduce the probability of analyzing routings generated by different protocols, it is preferable to learn protocols in data of short periods. In addition, to analyze the stability of routing protocols over time, inferred protocols from different periods of data can be compared.

Another reason for heterogeneity in the routings studied is that some routings are defined in real time by human operators, in order to route a specific call to a specific agent. The flexibility given by manual routings enables the system operators to deal with special situations which the programmed protocols fail to manage properly, for example when a customer asks to be served by the same agent that served him the last time he called.

This SBR-Mining work aims to learn only the ACD protocols, and not manual routings. In large call centers it is expected that only a small part of the routings executed in the system are done so manually.

Therefore logs of routings defined manually are undesired in the learning database, and should be separated from the ACD routings dataset. Nevertheless, usually routing logs do not state whether a routing was executed by the vertex protocol or by an operator, in which case the logs cannot be separated.

Data Quality

SBR-Mining leans on the quality of the database processed to infer the routing protocols. The desired database would include all the information processed by the ACD to route items, and a complete description of the items' flow.

One of the main qualities desired in the data is completeness, namely, the presence in the logs of all the information processed by the routing protocol. If event information that determines the preference score functions is missing, the method might fail to find good fitting protocols.

The second quality desired in the database is accurateness, specially of the timestamps. Inaccuracy in queue entry and exit timestamps may lead to incorrect skip samples, altering the train data.

The timestamp resolution has to be considered as well; when two events have the same timestamps it is not possible to learn whether one of the events' items had a bigger preference score.

In the studied cases, the timestamp resolution is one second, and there are pairs of items with similar timestamps. In these cases, we ignore the skip samples between the pair of items with the same timestamps.

4.3 Learning Routing Protocols—Three Examples

4.3.1 Intra-Vertex Protocol Learning (1) - Customer Vertex

We now use our intra-vertex algorithm to learn the routing protocol in the ILTelecom Business queue.

In Section 4.1, the ILTelecom system is described, and Figure 4.2 shows a realization of the system SBR flow. Figure 4.4 below shows the customers vertex studied in this example, and its neighboring vertices in the assumed design. Here we learn its intra-vertex protocol.

Iteration 1 - Step 1 - Define an intra-vertex score function p_v and its system state vector $x_t^{p_v}$ (as in Section 3.2.2, Formulation Part).:
We begin by studying skips under the FIFO protocol, where the preference score of a waiting item is its waiting time:

$$x_t^{p_v} = w_i(t),$$

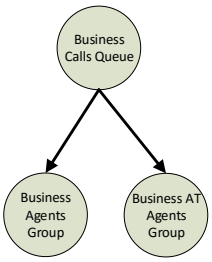

Figure 4.4: The graph represents the vertex of Business class customers studied in this example, and its SBR design.

$$p_v(x_t^{p_v}) = x_t^a = w_i(t),$$

where $w_i(t)$ is the waiting time of item i at time t ($i \in I_t^v$).

Iteration 1 - Step 2 - Compute skip variables for the sample of skip opportunities. Evaluate goodness-of-fit measures (as defined in Section 3.2.2, Testing Part)..
In this example, the goodness-of-fit is evaluated through the measures introduced in Section 3.2.2: the skipping item rate - the percent of items in the sample that skip out of those that could skip, and the skip rate - the percent of skips executed out of the number of skip opportunities in the sample. The skipping item rate value for the FIFO protocol is 23%, and the skip rate value is 9%. Clearly, the protocol in the sample differs from FIFO. Exploration is necessary to recognize routing patterns.

Iteration 1 - Step 3 - Build descriptive models of skip indicators and analyze skip explanatory variables (as described in Section 3.2.2, Exploration Part). Evaluate if the vertex can be forward decomposed into different item classes and can be split into new vertices. In this case return to Step 1 for each one of the new vertices..
To explore the reasons for the skips, we now test the following explaining vector:

$$x_t = (w_1(t), w_2(t), \Delta W_{i,j}(t), sub\_call_i, sub\_call_j, entry\_type_i, entry\_type_j),$$

where index i refers to the customer that can skip the other customer (j). The variables in the vector are the waiting time - $w_i(t)$, the difference between the waiting times - $\Delta W_{i,j}(t) = w_i(t) - w_j(t)$, the customer service number - sub_call_i, and the entry type - $entry\_type_i$ (a categorical value indicating the answer party type: VRU, Announce, Message, Directly).

To build the model, we use a decision tree algorithm [19, Chapter 9], since it builds interpretable models and, as was shown in [27], decision trees are good in identifying routing protocols. Some different values were tried for the meta-parameters running the algorithm, such as pruning criteria, minimum leaf size to split, etc.

Figure 4.5 shows the created decision tree model. The first variable splitting the tree is the customer subcall counter, indicating that customers that had previous service iterations in the system are prioritized to be served first. In addition, the model shows other splits based on waiting time variables.

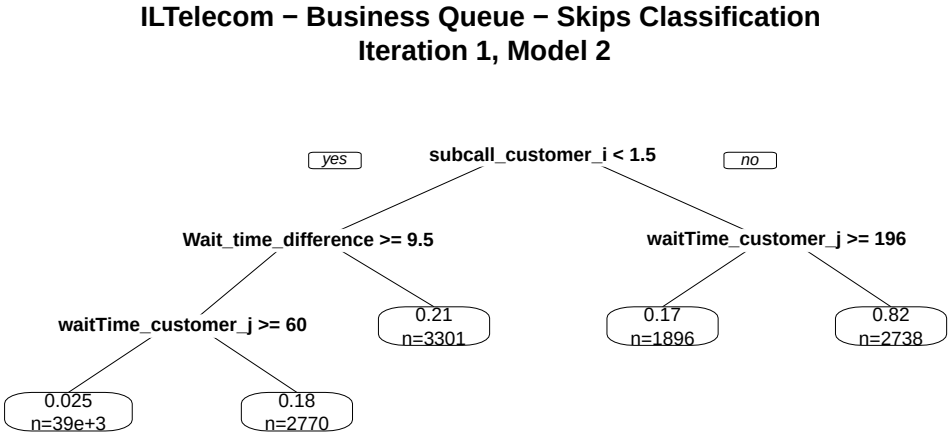

Figure 4.5: Decision tree model to explore violations of the FIFO protocol.

Iteration 1 - Step 4 Based on the insights from the analysis, define a new intra-vertex score function p_v and vector $x_t^{p_v}$ (as in Section 3.2.2, Formulation Part) and return to Step 2. If no insights are obtained, stop the learning process.

Since in the previous step it was found that customers which had previous service iteration tend to skip customers in the queue, we now formulate a vertex score function that boosts the score of customer proportionally to the number of previous service iterations. Formally,

$$x_t^{p_v} = (w_i(t), sub\_call_i),$$

$$p_v(x_t^{p_v}) = w_i(t) + sub\_call_i \cdot K_{SC},$$

where $w_i(t)$ and sub_call_i are as defined earlier and K_{SC} is a constant that represents the score added to a call for each previous subcall.

The K_{SC} value is to be selected so as to minimize the skip rate. Chart 4.6 shows that the number of skips decreases as the K_{SC} value grows, converging to the minimum number of skips when the K_{SC} exceeds 220.

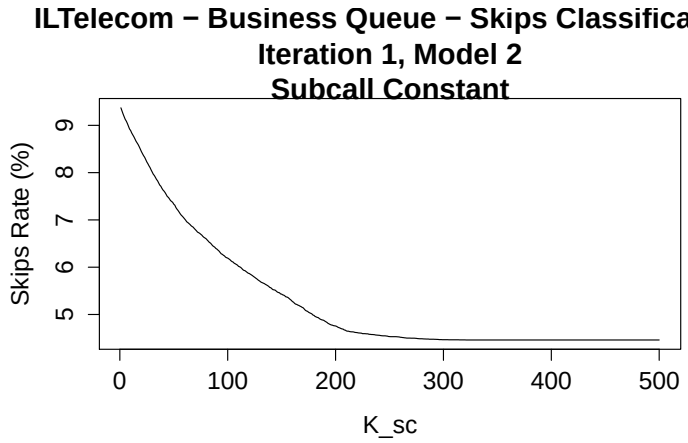

Figure 4.6: Goodness-of-fit of the vertex preference score function as a function of the subcall score constant K_{SC}.

Iteration 2 - Step 2 - Compute skip variables for the sample of skip opportunities. Evaluate goodness-of-fit measures (as defined in Section 3.2.2, Testing Part).
To evaluate the goodness-of-fit of the Vertex preference score, we compute the measures defined in the first iteration. The new protocol goodness-of-fit values are 19% for the skipping item rate and 4% skip rate. While the goodness-of-fit was significantly improved, we still continue the algorithm to seek a protocol with better goodness-of-fit values.

Iteration 2 - Step 3 - Build descriptive models of skip indicators and analyze skip explanatory variables (as described in Section 3.2.2, Exploration Part).
Evaluate if the vertex can be forward decomposed into different item classes and can be split into new vertices. In this case return to Step 1 for each one of the new vertices.
In this iteration we build a model using the explaining variables defined in the previous iteration, $x_t^{p_v} = (w_i(t), sub\_call_i)$, and the decision tree algorithm.

In the derived decision tree (Figure 4.7), the waiting time of Customer 1, the

customer with the lowest preference score value, is the first split variable. Its right branch shows that 976 skips were executed by customers that did not wait in the Business queue, skipping customers with the same sub_call value. The left leaves of the tree are all split by waiting time variables.

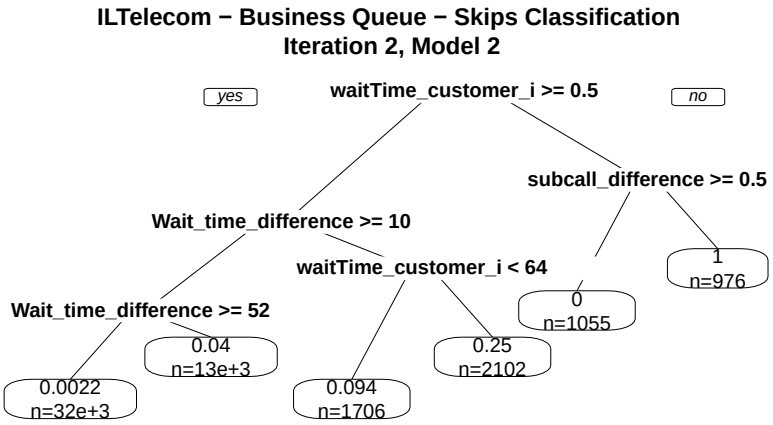

Figure 4.7: Decision tree model to explore violations of a protocol with preference score function: $p_v(x_t^{p_v}) = w_i(t) + sub\_call_i \cdot 220$.

Figure 4.8 illustrates skips as a function of the waiting time of pairs of customers in skip opportunities, customers i and j, where index i refers to the customer with the lower waiting time, and j to the other. The graph shows that several skips occur when the difference between customer waiting times is small, or immediately when the skipper customer arrives to queue (waiting time of customer i equal to zero). The skips that occur between customers with a small difference in the waiting time might be the result of inaccurate timestamps used by the routing protocol that generates the logs.

Iteration 2 - Step 4 - Based on the insights from the analysis, define a new intra-vertex score function p_v and vector $x_t^{p_v}$ (as in Section 3.2.2, Formulation Part) and return to Step 2. If no insights are obtained, stop the learning process.

To explore the skips where a customer skips immediately after arriving to queue, we explored variables based on the path of the customer since entry to the system and variables with information about the agent pools, such as which pool served the skipped call and the number of available agents in each pool. Unfortunately, this analysis did not lead to better score functions.

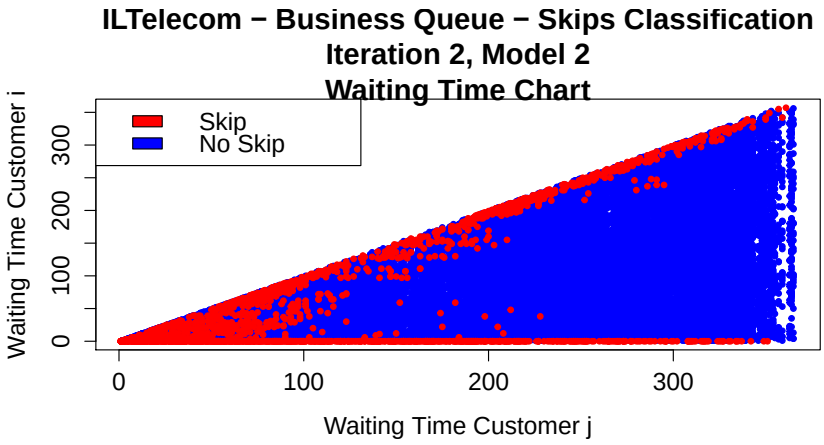

Figure 4.8: The graph represents customer waiting times from a sample of observed skip opportunities. Each blue/red point in the graph represents a skip opportunity.

If no improved score functions can be found, the vertex learning process is stopped at this step. We found that the intra-vertex protocol prioritizes customers according to the number of previous service interactions; the preference score function found is $p_v(x_t^{p_v}) = w_i(t) + sub\_call_i \cdot 220$. Comparing the initially assumed FIFO regime with the fitted protocol, we see that the goodness-of-fit was improved from a 9% skip rate and 23% skippers to a 4% skip rate and 19% skippers. to explain this modest improvement, recall that the initial protocol provided a good fit to start with.

4.3.2 Intra-Vertex Protocol Learning (2) - Agent Vertex

This example shows how the intra-vertex algorithm is used to learn the routing protocol of an agent vertex. In this case, the vertex is recognized to have two classes of agents which are split into two new vertices.

The example is computed on the EBO agents' pool of the USBank system. In Section 4.1, the USBank case is described, and Figure 4.1 shows a realization of the system's SBR flow[3]. Figure 4.9 shows the agent vertex studied in this example, and its neighboring vertices in the initial design. Here we study the learning of its intra-vertex protocol.

[3]Note that the EBO agents' group is called *Retail Agents' Group 9* in Figure 4.1, extracted from SEEgraph.

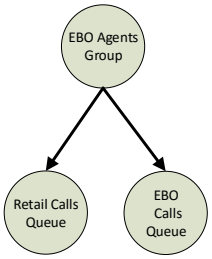

Figure 4.9: The graph represents the vertex of EBO agents studied in this example, and its initial assumed SBR design.

Iteration 1 - Step 1 - Define an intra-vertex score function p_v and its system state vector $x_t^{p_v}$ (as in Section 3.2.2, Formulation Part):
As in the previous example we begin studying skips under a FIFO protocol, for which the score function is the items' waiting time.

$$x_t^{p_v} = w_i(t),$$

$$p_v(x_t^{p_v}) = x_t^a = w_i(t),$$

where $w_i(t)$ is the waiting time of item i at time t. (Recall that waiting time for an agent is in fact idle-time: the time waiting to serve a customer).

Iteration 1 - Step 2 - Compute skip variables for the sample of skip opportunities. Evaluate goodness-of-fit measures (as defined in Section 3.2.2, Testing Part).
Goodness-of-fit is evaluated through the same measures defined in the previous example. The FIFO protocol skipping item rate is 35% and the skip rate is 24%. The values indicate a poor fitting of the FIFO protocol.

Iteration 1 - Step 3 - Build descriptive models of skip indicators and analyze skip explanatory variables (as described in Section 3.2.2, Exploration Part).
Evaluate if the vertex can be forward decomposed into different item classes and can be split into new vertices. In this case return to Step 1 for each one of the new vertices.
To explore skips, the following explaining vector is defined:

$$x_t = (w_i(t), w_j(t), node_i, node_j).$$

Index i refers to the customer that can skip the other customer (j). The variables w_i, w_j are the agents waiting, namely idle, time when the first agent is routed.

The variables $node_i, node_j$ are categorical variables indicating the agent nodes.

A decision tree model built for those explaining variables is illustrated in Figure 4.10. The first split is executed on criterion $w_i > 0.5$, creating two branches with skip rate 53% for observations that satisfy the constraint and 22% for the others. The succeeding splits are based on the node variables, showing a significantly higher skip rate when the agents originate from different nodes.

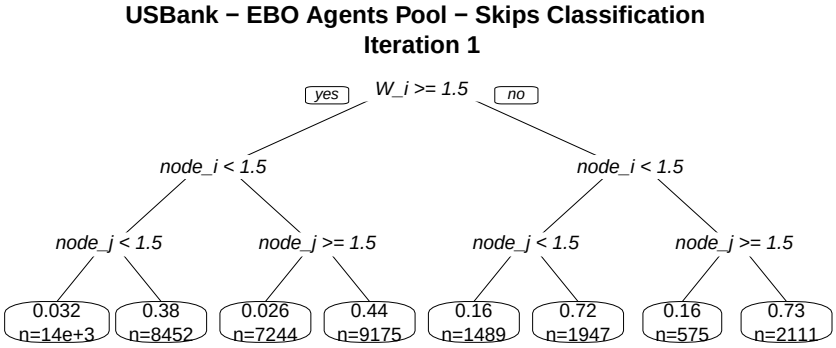

Figure 4.10: Decision tree to explore violations of a FIFO protocol.

The splits by the node variables suggests that the protocol routes agents from different nodes as different agent classes. Let us split the EBO vertex into two vertices, one for EBO agents in Node 1 and one for EBO agents in Node 2 (see Figure reffig:EBOExampleGraphSplit). Then new vertices' protocols will be learned.

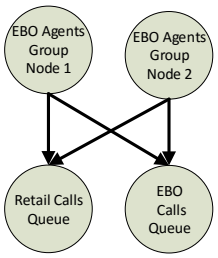

Figure 4.11: The graph represents the split vertex of EBO agents into two new vertices—one composed by the EBO agents in Node 1 and the other by the EBO agents in Node 2.

Iteration 2 - Step 1 - Define an intra-vertex score function p_v and its system state vector $x_t^{p_v}$. Compute skip variables for the skip opportunities sample:

Here we formulate the FIFO preference score function for both new vertices.

$$x_t^{p_v} = w_i(t),$$

$$p_v(x_t^{p_v}) = x_t^a = w_i(t).$$

Iteration 1 - Step 2 - Compute skip variables for the sample of skip opportunities. Evaluate goodness-of-fit measures (as defined in Section 3.2.2, Testing Part).
The goodness-of-fit values for the FIFO protocol in the EBO pool in Node 1 are skipping item rate 7% and skip rate 4%. For the pool in Node 2 the skipping item rate is 3% and skip rate 5%.

The FIFO protocol goodness-of-fit was improved from 35% of skipping agents and 24% of skips in the original vertex to 3–7% and 4–5% respectively in the new ones.

The application of our intra-vertex method led to the recognition of two different classes of agents in the original design definition. Once these classes were identified, their splitting into new vertices enables a better understanding of the protocol.

4.3.3 Inter-Vertex Protocol Learning

This example shows how the inter-vertex algorithm is applied to learn the protocol of an agent's vertex of the ILTelecom system (see description in Section 4.1). In this example, we learn the inter-vertex protocol of the vertex composed by the agents dedicated to serve customers of the class called Business AT (VIP Business calls). Figure 4.12 shows the SBR flow of the Business AT agents' vertex. As it is illustrated in the figure, agents from this vertex are routed to serve Business AT calls, and also regular Business calls. In this example, we learn the protocol that determines to which call vertices the Business AT agents are routed.

Iteration 1 - Step 1 - Define an inter-vertex score function P_v and its parameters $x_t^{P_v}$. (as in Section 3.2.3, Formulation Part).
In the first iteration, we test a protocol which routes the agents to the vertex with the longer queue length:

$$x_t^{P_v} = (Lq_{AT}, Lq_{BUS}),$$

$$P_v(x_t^{P_v}, v') = Lq_{v'}.$$

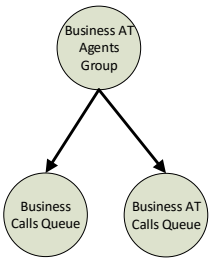

Figure 4.12: The graph represents the SBR flow of Business AT agents in the ILTelecom system.

Iteration 1 - Step 2 - Compute skip variables for the sample of skip opportunities. Evaluate goodness-of-fit measures (as defined in Section 3.2.3, Testing Part).
To evaluate the goodness-of-fit, we classify the sample observations into two sets: one for the skip opportunities where the Business Vertex has a higher score value and the AT vertex may skip it, and the other for the observations where the AT vertex has a higher score value and the Business vertex may skip it. The skip rate in the first set is 49%, showing that the AT Vertex skips the Business one under the tested protocol. In contrast, the skip rate of the second set is 2%, showing that the Business vertex does not skip AT. Clearly, further exploration is necessary to find how the score of the AT vertex is to be boosted.

Iteration 1 - Step 3 - Building descriptive models of skip indicators to analyze skip explanatory variables.
The analysis may be done separately for different subsets of skip opportunity observations (as described in Section 3.2.3, Exploration Part).

This exploration iteration focuses on the AT vertex skips. The selected sample is then the set of routing observations where the Business vertex has a higher preference score value, and its label is $Skips^{t}_{AT,BUS}$ indicating if the AT vertex skips the Business one or not.

Now, to find how we can formulate a new protocol that reduces the number of AT vertex skips, we seek to build a model where system state variables describe these skip values. To this end, we define a set of variables that we consider to be possible descriptive variables of the skips. The entries in the system state vector x_t are the number of items in each vertex (Lq_{AT} and Lq_{BUS}), the mean waiting

time of items in the vertices at the routing time (EW_{AT} and EW_{BUS}), and the maximum waiting time in the vertex (W_{AT} and W_{BUS}):

$$x_t = (Lq_{BUS}, Lq_{AT}, EW_{BUS}, EW_{AT}, W_{BUS}, W_{AT})$$

A decision tree algorithm was run to build a model describing the skip values through the defined vectors x_t. Figure 4.13 illustrates the computed tree, which consists of several splits based only on the maximum waiting time variables W_{BUS} and W_{AT}. It seems from the tree that the waiting time is one of the protocol parameters.

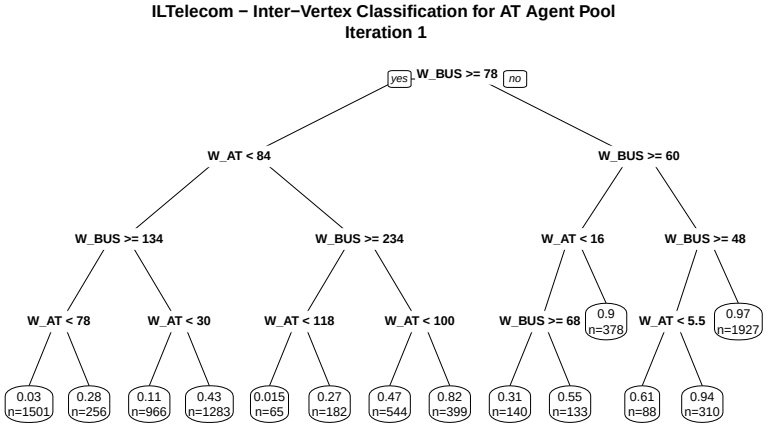

Figure 4.13: Decision tree model to explore violations of a protocol which routes the longer queue first.

Figure 4.14 plots the maximum waiting time in each vertex on the selected sample, and which vertex was chosen by the routing protocol. It can be seen in the plot that there exists an acceptable separation between AT and Business entries in terms of the waiting time variables. The plot shows four different areas, visually separated by straight lines. The first area is when the waiting time of the Business head of queue is under about 75 seconds. In this case only the AT vertex is selected. The second area is when the Business waiting time is between 75 to 150 seconds. In this case there exists a diagonal line separating areas where the Business or the AT vertex is selected. In the third area, the Business vertex head of queue waits above 150 seconds and below 200. In this case, the AT vertex is selected if its head of queue waiting time is above 100 seconds. In the last area, for higher waiting

times in the Business vertex, the selected vertex is determined by a diagonal line. In addition to the four areas, it can be noticed that the slopes of the lines in the first and third areas are close to zero while in the second and fourth line it is close to one.

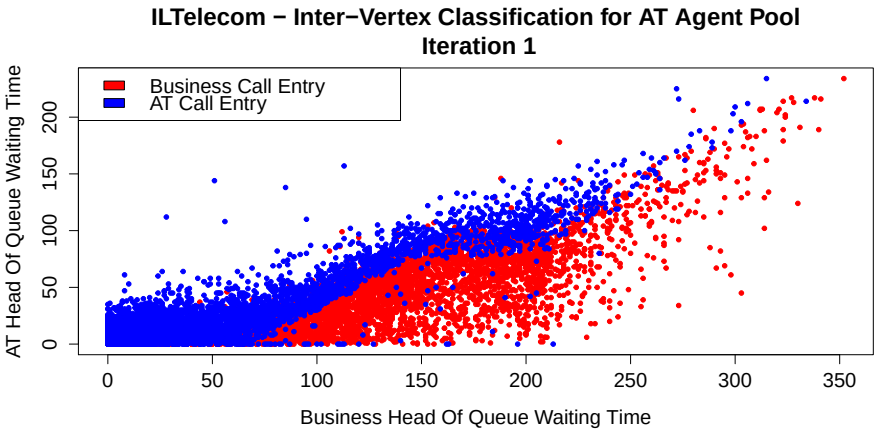

Figure 4.14: The graph represents the routing as a function of the waiting times of the AT and Business items in the head of the queues. Each point in the graph represents a routing instant, whose (x, y) values are the waiting times in the Business and AT queues respectively, and the color of the point indicates which queue was routed. Red points are Business routings and blue ones are AT queue routings.

In the exploration process, we learned that the maximum waiting time in each vertex is a good explaining variable of the routing decisions. In addition we learned how these variables are compared by the routing protocol. Below we apply the exploration insights to formulate a new protocol score function.

Iteration 1 - Step 4 - Based on the analysis insights, define a new inter-vertex score function P_v and vector $x_t^{P_v}$ (as in Section 3.2.3, Formulation Part) and return to Step 2. If no insights are obtained, stop the learning process.

We formulate a general preference score function where the four areas detailed above are defined by three constants, K_1, K_2, K_3, whose values are the axis x lengths of the first three areas. Then, the protocol is formulated as:

$$x_t^{P_v} = (W_{AT}, W_{BUS}),$$

$$p_v(x_t^{P_v}, v') = W_{v'} + I_{v'=AT} \cdot (K_1 + I_{W_{v'} > K_2} \cdot (K_3 - K_2)).$$

The exact values for the area lengths are computed to minimize the total number of skips in the sample. After numerical computation we found the optimal values: $K_1 = 63$, $K_2 = 92$ and $K_3 = 49$[4].

Iteration 2 - Step 2 - Compute skip variables for the sample of skip opportunities. Evaluate goodness-of-fit measures (as defined in Section 3.2.3, Testing Part). The skip rate of the AT vertex was reduced from 49% to 10% and the skip rate of the Business vertex was increased from 2% to 9%. The growth of the Business skips is expected under the new score function since it boosts the score of the AT vertex. The skip rates values are still high. In an additional exploration iteration we did not find new explaining variables to formulate a new protocol. We analyzed different variables, including the subcall number of customers in queues, total waiting times of items in previous subcalls, etc.

4.3.4 Results Summary

In our three examples, we identified some structures of routing protocols. In all cases, we found that the routing protocols are based on customer waiting times, but they are not pure FIFO policies.

In the example of ILTelecom intra-vertex, we found that the previous path of the customer influences its priority (previous subcalls), and in the USBank EBO agent vertices we found that agents are prioritized according to their geographical location (node). In the example of ILTelecom inter-vertex, where we studied how Regular and VIP Business customers are selected to be served by Business VIP agents, we found that the VIP customers are prioritized according to an index function of the customer waiting times.

Note that the initial goodness-of-fit value in the inter-vertex example is higher than the values in the intra-vertex examples. This difference is explained by the fact that in the first iteration of the learning process, the formulated score function usually ignores special attributes of items or vertices that can be selected. This kind of protocols, usually fit better real intra-vertex protocols that route homogeneous items, and do not fit well inter-vertex protocols that differentiates between specific vertices.

[4]The optimal K values were found computing the skip values of the formulated score function, for each set of K_1, K_2, K_3, for which $0 < K_1 < K_2 < K_3 < 300$. The skip values were computed in the sample of all the routing observations of the studied inter-vertex protocol.

In the three examples, the goodness-of-fit of the initially assumed protocols was improved by our learning method, but also after several learning iterations there still remained differences between the routing data and the estimated protocols.

We relate these differences to the issues described in Section 4.2, specifically data completeness and manual routings. The fact that most of the differences between the estimated protocols and the routing samples is when one of the items is served without waiting, which leads us to believe that these items have special attributes that are not stored in their logs, or they are manually routed.

Our results of the method application are summarized in Table 4.3 below:

Example	Initial Skip Rate	Initial Skipper Rate	Final Skip Rate	Final Skipper Rate
Example 1 - ILTelecom, Intra-Vertex Protocol (1)	9%	23%	4%	19%
Example 2 - USBank, Intra-Vertex Protocol (2)	24%	35%	3-7%	4-5%
Example 3 - ILTelecom, Inter-Vertex Protocol Note: results are computed for AT and Business call vertices respectively.	49%, 2%	-	10% , 9%	-

Table 4.3: Results Summary of SBR-Mining Application examples.

Chapter 5

Conclusions and Future Research

5.1 Conclusions

We developed a method for learning routing protocols in skill-based call centers. Specifically, the method consists of a general model for SBR systems, a mathematical formulation of routing protocols, a routing data model, a definition of protocol goodness-of-fit measures and finally, the learning method.

The SBR-Mining model is a network of agent pools and customer classes, assuming that the system design is known before the protocol learning. However, let us delve into the meaning of system design in the SBR-Mining context.

In the literature, design is characterized by groups of statistically independent and identically distributed items. From this characterization, we could derive two attributes of the design: 1. items in the same group can be routed to the same set of vertices, 2. there are no different priorities between items within the same vertex. In contrast to the common definition of SBR design, in real systems the design is commonly defined to adhere to the first of those attributes, but not the second. Therefore SBR-Mining assumes that the given SBR design defines the network flow, but not the priorities between items in each vertex. As part of the learning method, the homogeneity in the vertices is tested, and when it is found that one of the vertices is composed of more than one priority class, the vertex is split into new vertices and the network is updated.

One of the principal characteristics of our model is the duality between agent and customer vertices, which enables the use of the same learning method for both

kinds of vertices. This duality is present in the model and in the learning method structure, but in its application the differences between agent and customer routings must be considered: agent and customer event logs are different, their paths in the system are different and therefore the variables and protocols studied in the learning process must be different.

The mathematical formulation of the protocol is a function composed of intra- and inter-vertex protocols, and the protocol parameters are variables describing system states. We argued in Section 3.2 that the routing protocol functions are expected to be simple and intuitive, and that the challenge of SBR-Mining is in discovering the system state parameters. This argument is supported by the examples in Section 4, where we found simple routing protocols that explain more than 90% of the routing samples.

One of the challenges in developing our SBR-Mining method was the definition of the data model. Routings are selections of items to be routed, where the group of items that can be routed varies in time (and also in size). Therefore the definition of the routing sample structure is not straightforward. Previous work on learning routing protocols dealt only with inter-vertex protocols, where the group of vertices that can be routed is fixed.

The introduced data model defines a routing sample as a competition between a pair of items (or vertices) that wait at the same time to be routed by the same routing protocol. When one of the competing items is routed by the protocol, a routing decision is sampled. To learn the fit of the routing samples to estimated protocols we used the skip variables, which indicates whether the routing sample coincides with the estimated protocol selection.

To learn the routing protocols, our SBR-Mining runs classification algorithms to build a descriptive model of the skip variables, as a function of explanatory features that describe the competing items and the system state at the routing time. The method in Section 3.2 states that any classification algorithm could be applied in this stage, but in three case studies in Section 4.3 we found decision trees the most useful algorithms. The advantages of this algorithm is that it builds interpretable models from which it is simple to understand how the feature values lead to the model results.

Finally, SBR-Mining models the SBR protocol and suggests a learning method. The main limitation of the method is its iterative trial-and-error approach, as it

requires analysis of results in each learning iteration. These iterations could be perhaps, after some learning, executed automatically, having predefined sets of routing algorithms to test. The problem with such an automatic approach is that the predefined set of routing protocols must be adapted for each call center system according to its service process and design, and such an adaptation could require a similar analysis to that done in the iterative SBR-Mining method.

We applied our SBR-Mining method in 3 case studies. In all the cases, we learned routing protocols that improved the goodness-of-fit measures in comparison to the initially assumed protocols. Nevertheless, in all the cases, the learned protocols failed to describe all the routing samples. We offer two possible explanations for those differences: 1. There are features of routed items that are not stored in the studied logs, and 2. Manual routings executed by system managers, in order to respond to ad hoc system or customer requirements.

5.2 Future Research

Interesting worthwhile extensions are possible in the following directions:

- **Development of Automatic SBR-Mining Methods**
  The trial and error approach applied in this thesis could be extended to an automatic algorithm. Previous works in this field [4, 25] were limited to inter-vertex algorithms, used user-defined routing explanatory features and in some cases ignored process context data. An automatic algorithm may be able to deduce system-state variables from data logs, and apply machine learning algorithms to explain routing decisions. Such an algorithm could be based on the routing data model introduced in this work.

- **Skip Applications in Process Mining Conformance Analysis**
  A main focus of Process Mining is conformance analysis, which compares estimated process models with real process logs. Interesting future work could be to study applications of the skip measures in conformance analysis of routing and prioritization protocols. At present, there is a scarcity of Process Mining literature that studies protocol conformance. The skip variables could be the base of future research in this area.

- **SBR-Mining Method without the NIIT assumption**
  Our protocols learning assumes NIIT protocols, and therefore it does not learn how the protocols define the time when a routing must be executed. Extensions of SBR-Mining to protocols that do not satisfy NIIT could be investigated.

- **Estimated Protocols in System Simulations**
  Another interesting research direction is the application of system simulations in SBR-Mining. This application could compare real system performance, measured through QoS measures, and the system performance under estimated protocols. Such comparisons would enable measurement of estimated protocols goodness-of-fit under system targeted QoS measures. In addition, simulations based on estimated protocols could be used in the study of process performance under different "what-if" scenarios.

Bibliography

[1] Z. Akşin, M. Armony, and V. Mehrotra. The modern call center: A multi-disciplinary perspective on operations management research. *Production and Operations Management*, 16(6):665–688, 2007. 1

[2] Rami Atar, Avi Mandelbaum, and Gennady Shaikhet. Simplified control problems for multiclass many-server queueing systems. *Math. Oper. Res.*, 34(4):795–812, November 2009. 9

[3] S. L. Bell and R. J. Williams. On dynamic scheduling of a parallel server system with complete resource pooling. In *Analysis of Communication Networks: Call Centres, Traffic and Performance*, pages 49–71. American Mathematical Society, 2000. 12

[4] M. de Leoni, M. Dumas, and L. García-Bañuelos. Discovering branching conditions from business process execution logs. In *Fundamental Approaches to Software Engineering*, volume 7793 of *Lecture Notes in Computer Science*, pages 114–129. Springer Berlin Heidelberg, 2013. 18, 60

[5] SEE Center (Service Enterprise Engineering). `http://iew3.technion.ac.il/serveng/References` and `http://ie.technion.ac.il/Labs/Serveng`. 15, 37

[6] F. Folino, M. Guarascio, and L. Pontieri. Discovering context-aware models for predicting business process performances. In *On the Move to Meaningful Internet Systems: OTM 2012*, volume 7565 of *Lecture Notes in Computer Science*, pages 287–304. Springer Berlin Heidelberg, 2012. 15

[7] N. Gans, G. Koole, and A. Mandelbaum. Telephone call centers: Tutorial,

review, and research prospects. *Manufacturing & Service Operations Management*, 5(2):79–141, 2003. 1, 5, 6

[8] N. Gans and Y-P. Zhou. A call-routing problem with service-level constraints. *Operation Research*, 51(2):255–271, 2003. 12

[9] O. Garnet, A. Mandelbaum, and M. Reiman. Designing a call center with impatient customers. *Manufacturing & Service Operations Management*, 4(3):208–227, July 2002. 9

[10] O. Garnett. Introduction to skills-based routing and its operational complexities. Teaching note, Teachnion–Israel Insitute of Technology, 2000. 5

[11] J. C. Gittins. Bandit processes and dynamic allocation indices. *Journal of the Royal Statistical Society, Series B*, pages 148–177, 1979. 7

[12] E. S. Gordon. New problems in queues: Social injustice and server production management. PhD Thesis, Massachusetts Institute of Technology, 1987. 29

[13] I. Gurvich. Design and control of the m/m/n queue with multi-type customers and many servers. Msc. Thesis, Technion-Israel Institute of Technology, 2004. 5

[14] I. Gurvich, M. Armony, and A. Mandelbaum. Service-level differentiation in call centers with fully flexible servers. *Management Science*, 54(2):324–338, 2008. 12

[15] I. Gurvich, P. Liberman, and A. Mendelbaum. Skill based routing: Data-based review and research prospects. 2015. In preparation. 6

[16] I. Gurvich and W. Whitt. Queue-and-idleness-ratio controls in many-server service systems. *Mathematics of Operations Research*, 34(2):363–396, 2009. 10

[17] I. Gurvich and W. Whitt. Scheduling flexible servers with convex delay costs in many-server service systems. *Manufacturing & Service Operations Management*, 11(2):237–253, 2009. 9, 11

[18] I. Gurvich and W. Whitt. Service-level differentiation in many-server service systems via queue-ratio routing. *Operations Research*, 58(2):316–328, 2010. 11

[19] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, corrected edition, August 2009. 31, 46

[20] P. Liberman, A. Mandelbaum, and V. Trofimov. Skills-based-routing in usbank. Available at: http://http://ie.technion.ac.il/Labs/Serveng/files/Skills-Based-Routing_USBank.pdf, Technion–Israel Institute of Technology, Haifa, Israel, 2008. 39

[21] A. Mandelbaum and P. Momčilović. Performance-based routing. *Oper. Res. Lett.*, 42(6):418–423, September 2014. 13

[22] A. Mandelbaum and P. Momčilović. Personalized queues: The customer view, via least-patient-first routing. *Under revision to QUESTA*, 2014. 13

[23] A. Mandelbaum and A. Stolyar. Scheduling flexible servers with convex delay costs: heavy-traffic optimality of the generalized $c\mu$-rule. *Operation Research*, 52(6):836–855, 2004. 9

[24] A. Rozinat and W.M.P. van der Aalst. Conformance testing: Measuring the fit and appropriateness of event logs and process models. In *Proceedings of the Third International Conference on Business Process Management*, BPM'05, pages 163–176, Berlin, Heidelberg, 2006. Springer-Verlag. 14

[25] A. Rozinat and W.M.P. van der Aalst. Decision mining in prom. In *Business Process Management*, volume 4102 of *Lecture Notes in Computer Science*, pages 420–425. Springer Berlin Heidelberg, 2006. 18, 60

[26] C. Schaack and R. Larson. An N-server cutoff priority queue. *Operation Research*, 34(2):257–266, 1986. 12

[27] A. Senderovich, M. Weidlich, G. Avigdor, and A. Mandelbaum. Mining resource scheduling protocols. In *Business Process Management*, volume 8659 of *Lecture Notes in Computer Science*, pages 200–216. Springer International Publishing, 2014. 17, 19, 21, 31, 46

[28] A. Senderovich, M. Weidlich, G. Avigdor, and A. Mandelbaum. Queue mining – predicting delays in service processes. In *Advanced Information Systems Engineering*, volume 8484 of *Lecture Notes in Computer Science*, pages 42–57. Springer International Publishing, 2014. 17

[29] W.E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3(1-2):59–66, 1956. 9

[30] W.M. P. van der Aalst and S. Dustdar. Process mining put into context. *IEEE Internet Computing*, 16(1):82–86, 2012. 14

[31] W.M.P. van der Aalst. Business alignment: Using process mining as a tool for delta analysis and conformance testing. *Requir. Eng.*, 10(3):198–211, November 2005. 14

[32] W.M.P. van der Aalst, B.F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A.J.M.M. Weijters. Workflow mining: A survey of issues and approaches. *Data Knowl. Eng.*, 47(2):237–267, November 2003. 2, 14

[33] W.M.P. van der Aalst, T. Weijters, and L. Maruster. Workflow mining: Discovering process models from event logs. *IEEE Trans. on Knowl. and Data Eng.*, 16(9):1128–1142, September 2004. 2, 14

[34] J.A. Van Mieghem. Dynamic Scheduling with Convex Delay Costs: The Generalized c— mu Rule. *The Annals of Applied Probability*, 5(3):809–833, 1995. 9

[35] J.M.E.M. van der Werf, H.M.W. Verbeek, and W.M.P. van der Aalst. Context-Aware Compliance Checking. In *International Conference on Business Process Management (BPM 2012)*, volume 7481 of *Lecture Notes in Computer Science*, pages 98–113. Springer-Verlag, Berlin, 2012. 15