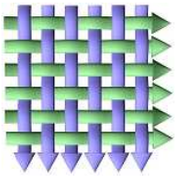
SFB 559 — Teilprojekt M1
LS Informatik IV
Universität Dortmund
23. Juli 2003

Sonderforschungsbereich 559

Modellierung großer
Netze in der Logistik

Technical Report 03010

ISSN 1612-1376



Input-Output Hidden Markov Models

for the Aggregation of Performance Models

SFB 559 — Teilprojekt M1

Falko Bause

Input-Output Hidden Markov Models for the Aggregation of Performance Models

Falko Bause
Informatik IV, Universität Dortmund
D-44221 Dortmund, Germany
falko.bause@udo.edu
July 23, 2003

Abstract— **This paper reports about experiments on generating aggregates for performance (sub)models. The aggregates are described by Input-Output Hidden Markov Models (IOHMMs). The parameters of the IOHMM are calculated from observation sequences obtained from the input/output behaviour of the original (sub)model.**

Keywords: Performance Evaluation, Aggregation, Input-Output Hidden Markov Models, Simulation, Optimisation

Contents

This research was supported by by the Deutsche Forschungsgemeinschaft as part of the Collaborative Research Centre "Modelling of large logistic networks"(559).

I. Introduction

It is well-known that analysis of models might be computationally intensive. Especially simulation of models of todays systems is extremely time-consuming. The use of (decomposition and) aggregation techniques might reduce the analysis effort. Aggregation is a well-known technique in the area of Queueing Networks (QN), since classes of models are known where this technique does not only reduce the analysis effort, but also leads to exact results [8].

The basic idea of aggregation can be described by the following procedure [7]:
1. Evaluate an isolated, detailed submodel separately.
2. Construct an equivalent, but less detailed substitute representation.
3. Use this substitute in place of the original submodel for purposes of evaluating the overall model.

The main problem in this process is to "construct an **equivalent** ... representation" of the submodel. In the QN setting a well-known type for such *equivalent* aggregates is the flow-equivalent server (FES), which serves customers with population-dependent speeds. The FES is determined by analysing the throughput of the short-circuited submodel for different populations. These population-dependent throughputs give the population-dependent service rates of the FES. For further details see [8]. This type of aggregate is very useful in the setting of product-form QNs, since one obtains exact results. In other environments only approximative performance results are computable. In most cases one can get only indications on the accuracy of the results by a series of experiments. Other types of aggregates, e.g. two-station mini-networks, which also lead to approximative results have been proposed in [7].

In this report we discuss a further idea how to build aggregates. We propose a new aggregate type based on Input-Output Hidden Markov Models [5].

The report is structured as follows. In Sect. II we define Input-Output Hidden Markov Models (IOHMMs). Sect. III explains how IOHMMs can be used for the aggregation of (sub)models. In Sect. IV we describe the optimisation procedure which has been used to determine an IOHMM aggregate and in Sect. V we demonstrate the capabilities of the approach by a variety of examples.

II. IOHMMs

An IOHMM[5] is a Markov Chain with state-transition probabilities dependent on input symbols. The behaviour of an IOHMM is as follows: The (hidden) Markov chain "reads" a symbol from a (given) input stream, changes its internal state and afterwards outputs a symbol both according to the read input symbol and dependent on each current state. The (hidden) Markov chain starts in initial state i according to an initial state distribution π_i at time $t = 0$.

The notion "hidden" is motivated from common applications of Hidden Markov Chains (IOHMMs with no input), especially in speech recognition. It is assumed that the observed output results are from a Markov Chain whose states are hidden to the observer. The main concern is to find a Markov Chain which best fits to the observed data.

We use the following notation for describing IOHMMs:

N	number of hidden states
Q	set of states $Q = \{1, \ldots, N\}$
M	number of symbols
V	set of symbols $V = \{1, \ldots, M\}$
q_t	state at step/time t, $t = 0, \ldots, T$
x_t	input symbol at step/time t, $t = 1, \ldots, T$
y_t	observed output symbol at time t, $t = 1, \ldots, T$
A	state-transition probability matrix with $a_{kij} = P[q_t = j \mid q_{t-1} = i, x_t = k)$
B	observation probability distribution with $b_{lj}(k) = P[y_t = k \mid q_t = j, x_t = l]$ i.e. the conditional probability of observing symbol k given the Markov process is in state j and the input symbol l is read
π	initial state distribution $\pi_i = P[q_0 = i]$

A describes the state-transitions of a Markov chain and thus the entries a_{kij} are independent of "time" t. This independence is also assumed for B. We assume that the set of states and symbols is finite, so that an integer encoding is sufficient.

$\lambda := (A, B, \pi)$ denotes the entire IOHMM.
Let $T \in \mathbb{N}$ denote the length of the observation sequence.
Matrices A and B have to satisfy the following conditions ("stochastic sub-matrices"):

$$\sum_{j=1}^{N} a_{kij} = 1, \forall k = 1, \ldots, M \text{ and } i = 1, \ldots, N \quad (1)$$

$$\sum_{k=1}^{M} b_{lj}(k) = 1, \forall l = 1, \ldots, M \text{ and } j = 1, \ldots, N \quad (2)$$

III. Defining Aggregates with IOHMMs

In this section we show how a template for IOHMM aggregates can be defined. For notation we use the *ProC/B* notation described in [4].

We assume that the following data has been collected from an appropriate observation of the (sub)model/system:

- a time interval $\Delta t \in \mathbb{R}^+$,
- an array $X = (x_1, \ldots, x_T)$ where $x_i \in \mathbb{N}_0$ denotes the number of arrivals at the sub-model in the i-th Δt time interval,
- an array $Y = (y_1, \ldots, y_T)$ where $y_i \in \mathbb{N}_0$ denotes the number of departures from the sub-model in the i-th Δt time interval,

Using some optimisation procedure, e.g. the Baum-Welch Algorithm (cf. Sect. IV), we assume that the following result has been determined:

- a state-transition probability matrix A,
- and an observation probability distribution described by a matrix B,

which hopefully describe the observed input/output of the (sub)model. The behaviour of an aggregate for this (sub)model can then be defined as follows:

The aggregate counts the number of arrivals in a time interval of length Δt. All arriving jobs are blocked in the aggregate, i.e. they are not allowed to leave. At the end of the time interval one step of the IOHMM is emulated giving an output value y for that time interval. This output symbol y can be interpreted as the number of "observed" departures. Thus, from the set of blocked jobs, y jobs are released and depart from the aggregate. If y exceeds the number of blocked jobs only the currently present number of jobs is released.

A more precise description of the IOHMM template, employing the *ProC/B* notation, is given in the next subsection.

A. Template IOHMM Aggregate

Fig. 1 shows the general form of an aggregate employing IOHMMs and the aggregation idea described in Sect. III. For simplicity this template aggregate only offers a single *Service* and serves a single class of customers. But the idea can also be extended to several services and multiple customer classes using an appropriate integer encoding for the IOHMM[1]. The service *Service* counts the number of arrivals and also keeps tracks of the currently number of blocked customers. The *max_stopped_procs* integer variable ensures that at most the number of blocked customers is released after receiving the "answer" from the functional unit (FU) modelling the behaviour of the IOHMM.

The second process chain in Fig. 1 models the control process which is started every *@@DeltaT* time units. This control process first calls the FU *IOHMM* which emulates one step of the IOHMM. The resultant output is interpreted as the number of customers which have to leave and the corresponding number of customers (at most the number of blocked customers) is released. Finally the control process resets the counter for arrivals.

Note that in our current implementation we used an integer encoding from $[1, \ldots, M]$ for the input and output symbols of our IOHMM. So, e.g., 1 encodes no arrival/departure, 2 encodes a single arrival/departure and in general i encodes $i-1$ arrivals/departures (see also the comment in Fig. 1).

Fig. 2 shows the model for the (template) IOHMM. Firstly, the input symbol is checked whether it is in the allowed range of symbols from 1 to M. Let us follow the *ELSE*-branch first, i.e. assume that the input symbol a is in $[1, \ldots, M]$. First a random number *randvar* between 0 and 1 is drawn from a uniform distribution and the appropriate successor state is calculated. This is done by repeatedly incrementing the variable j and adding the corresponding state change probability stored in matrix A. Once we exceed the value of *randvar* we select the current value of j as the new current state, which is stored in variable *current_state*. In a similar way we determine the output symbol employing matrix B.

It is a general design decision what we shall do if the input symbol is not in the assumed range of symbols. Note that we have determined this range by an observation of the original (sub)model. Whenever this aggregate is plugged into a different environment (or even in the original environment, if the aggregate does not behave identical to the original (sub)model), it might happen that we "see" an unknown input symbol, i.e. an unknown number of arrivals.
This design decision has to be based on the intended use of the aggregate. The design decisions in this paper are based on the assumptions that we are interested in an analysis of the system's **steady state** behaviour and that the original (sub)model does not create or destroy customers, i.e. follows a **"one-in-one-out behaviour"**. Following these assumptions the "flow in" of customers into the aggregate has to equal the "flow out". Assuming that the IOHMM aggregate will indirectly follow this "flow-in = flow out" condition concerning observed input and output symbols, it seems natural to define an identity mapping for unknown/unobserved input symbols. This simple extension gives us the possibility to use the IOHMM aggregate in different environments.

IV. Basic Algorithms for the determination of IOHMMs

In the next subsections we discuss algorithms trying to find an "optimum" IOHMM, which best reflects the observed behaviour, i.e. we are going to solve the following problem:
Given $X = (x_1, \ldots, x_T)$ *and* $Y = (y_1, \ldots, y_T)$,
estimate model parameters $\lambda = (A, B, \pi)$ *that maximise* $P[Y|X, \lambda]$.

Algorithms solving this optimisation problem for Hidden Markov Models are well-known (cf. [5], [9]).

A. A straight-forward, but inefficient approach

$P[Y|X, \lambda]$ can be determined directly from the definition of an IOHMM using the Markov property.

Let $q = (q_0, \ldots, q_T)$ be a state sequence. Since the observations are assumed to be independent, we have

$$P[Y|q, X, \lambda] = \prod_{t=1}^{T} P[y_t|q_t, x_t, \lambda] = \prod_{t=1}^{T} b_{x_t,q_t}(y_t)$$

The probability of a particular state sequence is given by

$$P[q|X, \lambda] = \pi_{q_0} a_{x_1,q_0,q_1} a_{x_2,q_1,q_2} \ldots = \pi_{q_0} \prod_{t=1}^{T} a_{x_t,q_{t-1},q_t}$$

Note that we have

$$P[Y, q|X, \lambda] = P[Y|q, X, \lambda] * P[q|X, \lambda]$$

[1] E.g., the input symbol **1**0702**2**0800 might encode the "arrival" of 7 class-1 customers and 2 class-2 customers having called service**1** in the observed time interval and 8 class-1 and 0 class-2 customers having called service**2**, assuming that at most 99 customers of each class will arrive.

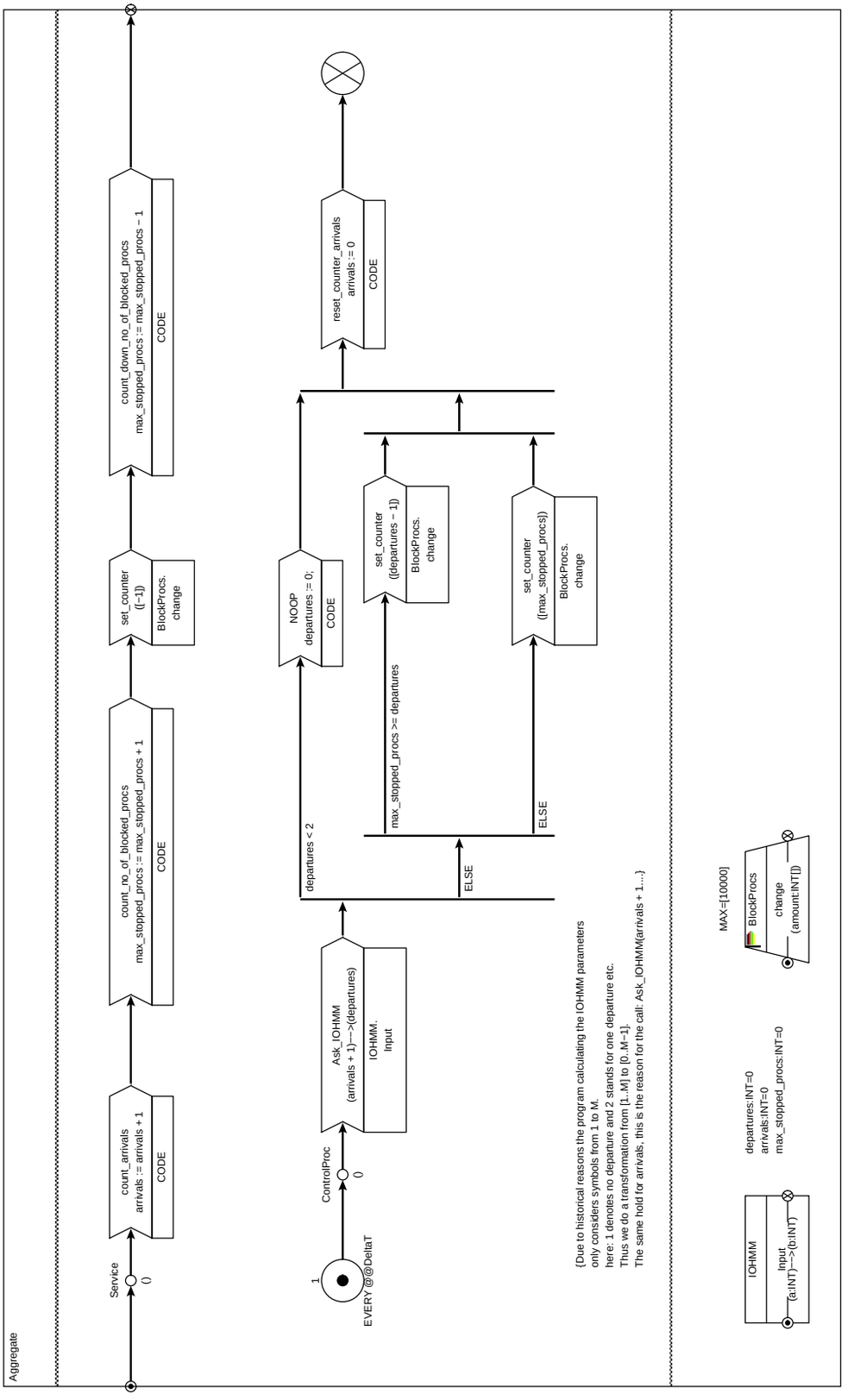


Fig. 1. Template IOHMM Aggregate

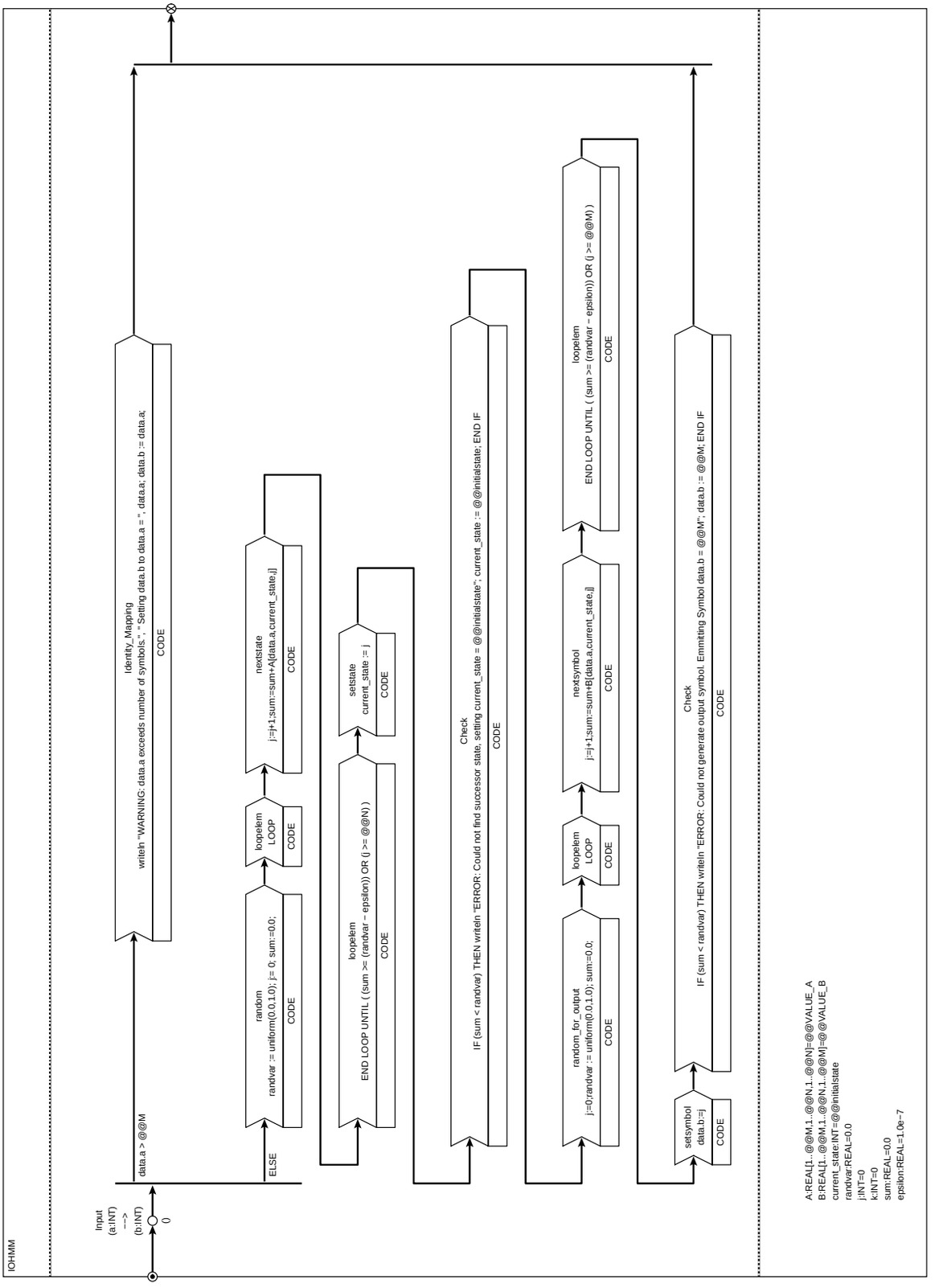


Fig. 2. Template IOHMM

which finally gives

$$P[Y|X,\lambda] = \sum_{\text{over all sequences } q} P[Y,q|X,\lambda]$$

Unfortunately the worst-case time complexity of this approach is in $O(TN^T)$. The Baum-Welch-Algorithm, described later, is a more efficient alternative. Before describing this algorithm, we will present two algorithms which are used in the Baum-Welch-Algorithm.

B. Forward-Algorithm

The Forward-Algorithm calculates $\alpha_t(i)$, the probability to observe the sequence $y_1,\ldots,y_t$, such that the state at time t ($t = 1,\ldots,T$) is i (i.e. $q_t = i$) given the input sequence $x_1,\ldots,x_t$ and the model parameters λ. The worst-case time complexity of this algorithm is in $O(N^2T)$.

$$\alpha_t(i) := P[y_1,\ldots,y_t,q_t = i|x_1,\ldots,x_t,\lambda],$$

$t = 1,\ldots,T$.
Initialisation: Define the auxiliary function

$$\alpha_0(i) := \pi_i$$

Induction for $t = 0,\ldots,T-1$:

$$\alpha_{t+1}(j) = [\sum_{i=1}^{N} \alpha_t(i) a_{x_{t+1},i,j}] b_{x_{t+1},j}(y_{t+1})$$

Termination:

$$P[Y|X,\lambda] = \sum_{i=1}^{N} \alpha_T(i)$$

With that the probability, which we want to maximise, can be calculated more efficiently than using the first approach.

C. Backward-Algorithm

The Backward-Algorithm is also used as a subroutine in the Baum-Welch-Algorithm.

The Backward-Algorithm calculates $\beta_t(i), t = 0,\ldots,T-1$, the probability to observe the sequence $y_{t+1},\ldots,y_T$, given the state at time t is i (i.e. $q_t = i$), the input sequence is $x_{t+1},\ldots,x_T$ and the model parameters λ. The worst-case time complexity of this algorithm is also in $O(N^2T)$.

$$\beta_t(i) := P[y_{t+1},\ldots,y_T|q_t = i, x_{t+1},\ldots,x_T,\lambda],$$

$t = 0,\ldots,T-1$.

Initialisation: Define the auxiliary function

$$\beta_T(i) := 1, i = 1,\ldots,N$$

Induction for $1 \le i \le N, t = T-1,\ldots,0$:

$$\beta_t(i) = \sum_{j=1}^{N} a_{x_{t+1},i,j} b_{x_{t+1},j}(y_{t+1}) \beta_{t+1}(j)$$

D. Baum-Welch-Algorithm

The Baum-Welch-Algorithm iteratively determines a sequence of model parameters $\lambda_i, i = 1,\ldots$. It can be shown that the sequence of probabilities $P[Y|X,\lambda_i], i = 1,\ldots$ does not decrease and thus a usual iteration scheme can be defined. In the following we describe on iteration step of the Baum-Welch-Algorithm, i.e. how to determine λ_{i+1} given λ_i.

Define $\xi_t(i,j,k)$ as the probability of being in state i at time t and in state j at time $t+1$, given X and Y, where k denotes the input symbol at time $t+1$, i.e.

$$\xi_t(i,j,k) := \begin{cases} \frac{\alpha_t(i) a_{k,i,j} b_{k,j}(y_{t+1}) \beta_{t+1}(j)}{P[Y|X,\lambda]} & \text{if } x_{t+1} = k \\ 0 & \text{otherwise} \end{cases}$$

for $t = 0,\ldots,T-1$.
Define $\gamma_t(i,k)$ as the probability of being in state i at time t, given X and Y, where k denotes the input symbol at time $t+1$, i.e.

$$\gamma_t(i,k) = \sum_{j=1}^{N} \xi_t(i,j,k), \qquad t = 0,\ldots,T-1$$

With these definitions we have
$\sum_{t=0}^{T-1} \gamma_t(i,x_{t+1})$ is the expected number of times state i is visited, given X and Y,
$\sum_{t=0}^{T-1} \xi_t(i,j,x_{t+1})$ is the expected number of transitions from state i to state j, given X and Y.

Define new model parameters $\lambda_{i+1} := \bar{\lambda} := (\bar{A},\bar{B},\bar{\pi})$ by

$\bar{\pi}$:

$$\bar{\pi}_i := \gamma_0(i,x_1)$$

$\bar{A}$:

$$\begin{aligned} \bar{a}_{kij} &= \frac{\text{exp. \#transitions from } i \text{ to } j \text{ for inp. symbol } k}{\text{exp. \#transitions out of } i \text{ for inp. symbol } k} \\ &= \frac{\sum_{t=0}^{T-1} \xi_t(i,j,k)}{\sum_{t=0}^{T-1} \gamma_t(i,k)} \end{aligned}$$

$\bar{B}$:

$$\begin{aligned}\bar{b}_{lj}(k) &= \frac{\text{exp. \#outputs } k \text{ in } j \text{ for inp. symbol } l}{\text{exp. number to be in } j \text{ for inp. symbol } l} \\ &= \frac{\sum_{\substack{t=0 \\ y_{t+1}=k}}^{T-1} \gamma_t(j,l)}{\sum_{t=0}^{T-1} \gamma_t(j,l)}\end{aligned}$$

Very often the Baum-Welch-Algorithm only finds a local optimum (cf. [9]). It is also well known that the algorithm tends to change significantly more entries of matrix B than elements of matrix A. In order to get a better optimisation procedure we combined the Baum-Welch-Algorithm with a simple $(1,1)$ evolutionary strategy (cf. [11]):

After a fixed number of iterations of the Baum-Welch-Algorithm (10 in our examples[2]) we randomly changed entries of matrix A using the following algorithms:

```
EA_LEVEL = 0.9; /* fixed for our experiments */
randvar = drand48(); /* random number in [0,1] */

while (randvar <= EA_LEVEL)
 {
  changeentries(A);
  randvar = drand48();
 }
```

The number of calls to *changeentries* is thus governed by a Bernoulli experiment. Procedure *changeentries* is depicted in Fig. 3 and does the following, explained in plain words: for each input symbol and state we select two successor states randomly (i.e. we select two matrix entries for fixed k,i) and change the corresponding matrix entries. The entries are changed according to a random value *randvar* such that (1) remains valid. Furthermore *randvar* is chosen such that the initial structure of the hidden Markov chain is not changed, i.e. we do not allow creation or deletion of state transitions.

In the following subsection we report on results from a series of experiments.

Some remarks on the implementation:
We first implemented the described algorithm in C. During our experiments we also used long observation sequences (about 1000*) which resulted in very small numbers for the probabilities* $P[Y|X,\lambda]$*. The number representation of the used hard- and software (2 GB RAM SunBlade 100 running SunOS 5.8) gives a range from 1.797693e+308 ("max_normal") to 2.225074e-308 ("min_normal"). As several results for long observations (*$T \approx 1000$*) indicate (e.g. see Table III) this default number representation is insufficient and even for those cases where we can represent the path probabilities, we had to struggle with numerical instability due to the small numbers. Thus we decided to enhance the implementation by the arbitrary precision library MAPM [12] using a minimum of* 50 *digits for all computations. As one might imagine this increased the computation time significantly.*

[2]In most cases, Baum-Welch gets stuck in a local optimum after a few number of iteration steps.

```
for (k = 1; k <= M; k++)
 for (i = 1; i <= N; i++)
 { /* Select two entries randomly and
      change entries according
      to a random value,
      such that the sum of A[k,i,j]
      over all j is still 1.0 */
      j1 = 1; j2 = 1;
  while (j1 == j2)
  {
   randvar = drand48();
   j1 = (int)(randvar * N) + 1;
   randvar = drand48();
   j2 = (int)(randvar * N) + 1;
  }
  /* Do not change entries
     which are approx. 0 or 1 */
  if ((A[(k,i,j1)] > EPSILON) &&
      ((1 - A[(k,i,j2)]) > EPSILON))
  {
   /* Determine random value
      for entry changes of
      A[k,i,j1] and A[k,i,j2]
      and change values */
      randvar = drand48();
  while ((randvar >= (A[(k,i,j1)])) ||
         (randvar >= ((1-A[(k,i,j2)]))))
  {  randvar = drand48();
     randvar  *=
         min(A[(k,i,j1)],1-A[(k,i,j2)]);
  }
  /* Adding and substracting randvar
     leaves A sub-stochastic */
  A[(k,i,j1)] -= randvar;
  A[(k,i,j2)] += randvar;
 }
```

Fig. 3. Procedure *changeentries*

V. First Experiments

We first started with a simple experiment "aggregating" a M/M/1 queue.

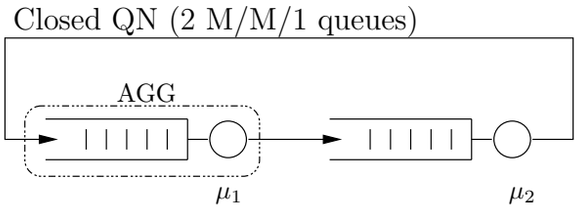


Fig. 4. Closed Tandem Queueing Network

Fig. 4 depicts the original model. The model is a closed queueing network comprising two M/M/1 queues. The network is a product-form queueing network and performance figures can be calculated efficiently, e.g. employing Mean Value Analysis (MVA).

For a first experiment we selected the following set of parameters: $\#Jobs = 5, \mu_1 = 1.2, \mu_2 = 1.3$ giving the results shown in Table II.

queue	population	throughput	sojourn time
1	2.732541	1.037791	2.633055
2	2.267460	1.037791	2.184891

TABLE II
Results of the QN of Fig. 4 for
$\#Jobs = 5, \mu_1 = 1.2, \mu_2 = 1.3$

We selected queue AGG for "aggregation" and queue 2 for validation. In particular we did the following:

1. We simulated the QN of Fig. 4 for the given parameter set. The simulator code was augmented by additional on-the-fly outputs of the form

```
...
ARRIVAL     5.016166761024598130802587E+003   1
DEPARTURE   5.018955154526076434251535E+003   1
ARRIVAL     5.019071368393921872552709E+003   1
DEPARTURE   5.019827164277301356776206E+003   1
DEPARTURE   5.020005177702461907074393E+003   1
ARRIVAL     5.020196253371342898219609E+003   1
...
```

recording the arrival and departure events at queue AGG. The first column specifies whether an arrival or a departure happened, the second column denotes the model time at which the corresponding event had happened. The third column entry gives the number of arrivals/departures at the specified model time. Note that in the queueing network of Fig. 4 only single arrivals/departures take place.

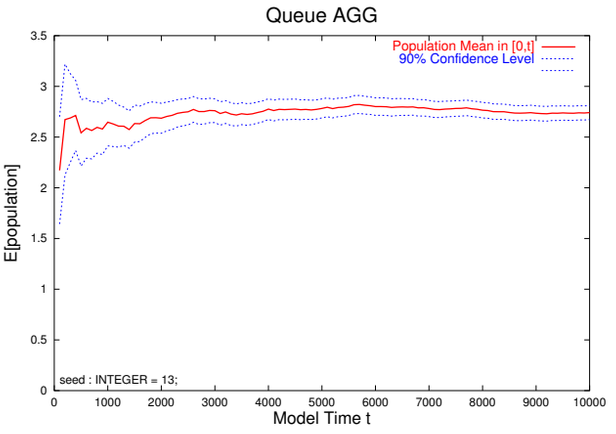


Fig. 5. Population at queue AGG

This output was collected after a long warm-up phase of 5000 model time units thus trying to make sure that the output information belongs to the steady-state phase of the model. E.g., Fig. 5 ($\#Jobs = 5, \mu_1 = 1.2, \mu_2 = 1.3$) shows the mean number of customers at queue AGG illustrating the short initial transient phase of the model. A point at time t of the shown curve denotes the mean number of customers at queue AGG for the time interval $[0, t]$.

2. The simulation output was afterwards transformed (for a given time interval Δt and observation length T) to input values ($X = (x_1, \ldots, x_T)$ and $Y = (y_1, \ldots, y_T)$) for the optimisation procedure described in Sect. IV giving an IOHMM specification. The number of symbols, M, was always determined by $M = max_{i=1}^{T}(max(x_i, y_i))$. The initial setting for matrices A, B and the initial probability distribution π was 1 for each entry followed by a normalisation step in order to ensure proper probability distribution definitions (cf. Eq. (1) and (2)).

3. Finally, we substituted queue AGG by the template IOHMM aggregate (cf. Sect. III-A) plugging in the calculated IOHMM specification. The resultant model was simulated, giving results for queue 2, see, e.g., Table III.

The columns of Table III denote the following:
Δt is the Δt time interval (cf. with "@@DeltaT" in Fig. 1),
N denotes the number of states,
T denotes the length of the observation sequence
population is the mean of the population at queue 2 after simulating 10000 time units,

Δt	N	T	population	throughput	sojourn time	Prob. start	Prob. end	#iter
EXACT			**2.267**	**1.038**	**2.185**			
0.8	6	1002	1.82	0.93	1.96	7.8E-499	6.2E-483	241
0.8	15	202	1.76	0.93	1.90	3.6E-98	3.7E-98	71
0.8	15	1002	1.72	0.91	1.90	7.8E-499	7.8E-497	141
0.3	6	102	1.31	0.80	1.64	1.6E-26	1.8E-26	80
0.3	6	1003	1.72	0.93	1.85	2.1E-296	4.6E-296	74
0.3	10	90	1.98	0.94	2.10	3.7E-24	7.7E-19	330
* 0.3	10	1003	2.21	1.04	2.13	4.7E-327	4.9E-327	118
0.3	15	102	1.73	0.90	1.93	5.6E-29	3.6E-25	119
* 0.3	15	1003	2.20	1.03	2.12	4.7E-327	6.1E-327	65
* 0.3	25	1003	2.20	1.04	2.13	4.720E-327	4.728E-327	14
0.1	6	1004	1.87	0.94	1.99	1.5E-140	2.1E-140	100
0.1	15	1004	1.85	0.94	1.97	1.50E-140	1.51E-140	60
0.1	15	3003	1.79	0.93	1.91	2.3E-437	2.4E-437	62
0.1	30	277	1.51	0.86	1.76	5.98E-35	5.98E-35	9
0.1	45	3003	takes too long, no optimisation result			2.26E-437	2.26E-437	24

TABLE III

RESULTS (MEAN VALUES) FOR QUEUE 2 IN THE MODEL WITH SUBSTITUTED QUEUE AGG ($\#Jobs = 5, \mu_2 = 1.3$)

throughput is the mean of the throughput at queue 2 after simulating 10000 time units,
sojourn time is the mean of the sojourn time at queue 2 after simulating 10000 time units,
Prob. start denotes $P[Y|X, \lambda]$ after the first 10 iterations of the optimisation procedure described in Sect. IV. Note that E/A-steps only occur after 10 iteration steps of the Baum-Welch algorithm, i.e. no E/A-step has been performed in the first 10 steps. Numbers are represented by an exponential notation, e.g. 7.8E-499 denotes the number $7.8 * 10^{-499}$.
Prob. end denotes $P[Y|X, \lambda]$ when we terminated the optimisation procedure.
#iter denotes the overall number of Baum-Welch iteration steps. #iter does not count/include any E/A-step.

The theoretically exact values are shown in the first line. They have been determined using the MVA algorithm for product-form QNs [8]. All result values (*population, throughput, sojourn time*) have been rounded to 2 digits after the decimal point. In further tables we will switch to rounding to 3 digits.

Most results depicted in Table III show a significant difference to the exact values. Only those lines marked by an asterisk give satisfactory results, keeping in mind the very low complexity of our example. Surprisingly, columns *Prob. start* and *Prob. end* indicate that we did not improve much during the iterations of the optimisation procedure. Furthermore, due to long running times, only a very few number of iterations could be performed for larger state values, N. Two reasons might explain the shown results: First, the choice of values for $\Delta t, N$ and T has a significant impact on the overall quality of the aggregate or, second, by chance all optimisation results are bad except the ones marked with an asterisk. We decided that the first reason seems more plausible and did further experiments with other parameter values.

Before starting with a new round of experiments, let us first have a closer look to the example tandem QN. Table IV shows more theoretical results. Leaving a state (n_1, n_2) implies a departure in our special example. Thus the mean time being in a state (except state $n_1 = 0$) conforms to the mean departure time of our model. Comparing Tables III and IV we conjecture that we get better results if Δt is close to the average time leaving a state (see "time in state (n_1, n_2)" in Table IV). The average time in all states, $s \in S$, is given here by $\sum_{s \in S}(\text{time in state}(n_1, n_2)\pi(n_1, n_2)) = 0.537$.

With that conjecture we started various experiments also with different service rates for queue AGG (results are shown in Table V), having in mind theoretical results shown in Table VI. Unfortunately, the results are still far from giving good aggregates based on IOHMMs.

As we see from the results, the optimisation might take too long. Since the running time for the optimisation procedure depends quadratically on the number

n_1	0	1	2	3	4	5
$\pi(n_1, n_2)$	0.135	0.146	0.159	0.172	0.186	0.202
time in state (n_1, n_2)	0.77	0.4	0.4	0.4	0.4	0.83
mean time between to consecutive visits to state (n_1, n_2) = $\frac{\text{Time in } (n_1, n_2)}{\pi(n_1, n_2)}$	5.69	2.73	2.52	2.33	2.15	4.13

TABLE IV
RESULTS OF THE QN OF FIG. 4 FOR $\#Jobs = 5, \mu_1 = 1.2, \mu_2 = 1.3, n_2 = 5 - n_1$

μ_1	μ_2	Δt	N	T	population	throughput	sojourn time	Prob. start	Prob. end	#iter
1.2	1.3	0.537	10	502	1.968	0.974	2.020	1.8E-209	4.1E-186	960
1.2	1.3	0.537	10	1003	1.764	0.933	1.891	2.5E-410	2.5E-410	501
1.2	1.3	0.4	10	502	1.696	0.920	1.844	2.3E-173	2.3E-173	132
1.2	1.3	0.4	10	1002	1.749	0.942	1.856	3.2E-355	4.6E-355	838
EXACT					**2.267**	**1.038**	**2.185**			
0.9	1.3	0.7	10	502	1.448	0.817	1.773	4.4E-222	3.5E-201	589
0.9	1.3	0.7	15	502	inappropriate optimisation result			4.4E-222	4.4E-222	24
EXACT					**1.508**	**0.851**	**1.773**			
1.8	1.3	0.48	10	503	2.189	0.996	2.198	2.2E-215	5.1E-203	1089
EXACT					**3.392**	**1.217**	**2.787**			

TABLE V
RESULTS (MEAN VALUES) FOR QUEUE 2 IN THE MODEL WITH SUBSTITUTED QUEUE AGG ($\#Jobs = 5$)

n_1	0	1	2	3	4	5
$\mu_1 = 0.9, \mu_2 = 1.3,$ **average time in states** $= 0.699$						
$\pi(n_1, n_2)$	0.055	0.079	0.115	0.166	0.239	0.346
time in state (n_1, n_2)	0.769	0.455	0.455	0.455	0.455	1.111
mean time between to consecutive visits to state (n_1, n_2) = $\frac{\text{Time in } (n_1, n_2)}{\pi(n_1, n_2)}$	13.989	5.723	3.962	2.743	1.899	3.214
$\mu_1 = 1.8, \mu_2 = 1.3,$ **average time in states** $= 0.482$						
$\pi(n_1, n_2)$	0.324	0.234	0.169	0.122	0.088	0.064
time in state (n_1, n_2)	0.769	0.323	0.323	0.323	0.323	0.556
mean time between to consecutive visits to state (n_1, n_2) = $\frac{\text{Time in } (n_1, n_2)}{\pi(n_1, n_2)}$	2.376	1.380	1.910	2.645	3.663	8.734

TABLE VI
FURTHER RESULTS OF THE QN OF FIG. 4 FOR $\#Jobs = 5, n_2 = 5 - n_1$

of state, we decide to reduce the complexity of the overall optimisation and start experiments with a reduced number of jobs, setting $\#Jobs = 3$. We hope that this "size of the problem" is not too trivial, still giving us the chance to get some insight into the effects of parameter settings giving "good" aggregates, but that the problem size is small enough to be handled by our prototype implementation of the optimisation algorithm. Table VII presents the results from a series of experiments. The choice of Δt is guided

by the theoretical results shown in Table VIII. Again the results are far from being convincing. Especially the result values for the case $\mu_1 = 0.1$ are suspicious, since the quality gets worse with increasing T.

All the shown results indicate that the proposed aggregate (cf. Fig. 1) nearly always gives results smaller than the theoretical values. Having a closer look at our template aggregate we realise that if only a few jobs/customers are blocked we will discard information from the IOHMM by releasing only the number of blocked processes. Since we assume that the IOHMM respects the "flow in = flow out" condition, this construct will violate the "flow in = flow out" for the overall aggregate. The current aggregate will then probably "block" more customers than the corresponding original (sub)model, which also explains the low population and other values for the non-aggregated queue 2. In the next section we will revise our template aggregate and redo some of the above described experiments in the now changed setting.

VI. Revised Definition of Aggregates employing IOHMMs

As mentioned in the last section we changed the *Aggregate* definition of the aggregate of Fig. 1. The new aggregate is shown in Fig. 6

The new aggregate differs from the former one in the definition of the control process. Irrespective of the number of currently blocked processes, the counter *BlockProcs* is always set to *departures - 1.* A call to service *Service* might thus result in no blocking of the calling customer, so that the customer leaves the aggregate in **zero time**.

VII. Further Experiments

We used the IOHMM definitions obtained from the experiments described in Sect. V for new simulation experiments. The results are shown in Table X. The last two columns denote the following:

sum of all differences *:*
(departures - 1) - max_stopped_procs
Whenever the IOHMM "answer" exceeds the the number of blocked processes, we stored this value. The result shown, is the sum of all these positive values for the total simulation run of 10000 model time units.

sum of usage of identity mapping *:*
(= sum of (inp. symbol - M))
Whenever the IOHMM is "asked" using an unknown input symbol, i.e. an input symbol which has not been encountered as an input symbol during the optimisation procedure, we stored this value minus the M, since M is maximum input value encountered during the optimisation procedure. The result shown, is the sum of all these positive values for the total simulation run of 10000 model time units.

The results show that also this form of aggregate has its deficits and the performance figures do only conform to the original ones in very few cases. Now we overestimate the theoretical values, which might be caused by "service" of customers in zero time. Having a closer look at the second last column ("sum of all differences *(departures - 1) - max_stopped_procs* ") we realise that this difference might be caused by the different views both functional units have on the "state" of the overall aggregate. The FU modelling the *IOHMM* has some internal state ("current_state") and calculates a corresponding response. Since this response might not conform to the number of blocked processes we adjust the answer of the IOHMM somehow. But this adjustment does not influence the internal state of the FU *IOHMM*. So it might happen that the internal state of the IOHMM and the "real" state the aggregate differ more and more as time goes by.

VIII. Revised Definition of FU IOHMM and Experiments

For a next series of experiments we decide to redesign the FU *IOHMM* in such a way that this divergence in the "real state" of the aggregate and the "IOHMM internal knowledge" on the state do not diverge. The simplest such way is to accept only that output of the IOHMM which conforms to the number of blocked processes. I.e. we do only accept those "answers" of the IOHMM which do not exceed the number of currently blocked processes. The modified definition of the FU *IOHMM* is shown in Fig. 7. With this definition of the FU *IOHMM* both definitions of the aggregate (i.e. FU *IOHMM Aggregate*; cf. Figs. 1 and 6) result in the "same behaviour".

The results of some experiments with this revised definition of FU IOHMM are shown in Table XI. Still the results are non-satisfactory.

IX. Further Redefinition of FU Aggregate and Experiments

For a next series of experiments we follow [3] and decide to redesign the FU *Aggregate* again and keeping the old (i.e. the first) version of the definition of FU IOHMM. As mentioned this might give us some

μ_1	μ_2	Δt	N	T	population	throughput	sojourn time	Prob. start	Prob. end	#iter
0.1	1.3	9.0	5	252	0.088	0.109	0.811	2.8E-55	9.1E-48	221
0.1	1.3	9.0	5	502	0.502	0.334	1.504	2.6E-98	1.4E-82	175
EXACT					**0.083**	**0.099**	**0.832**			
0.9	1.3	0.75	5	502	0.961	0.740	1.298	6.0E-210	3.9E-197	213
0.9	1.3	0.75	5	1005	0.939	0.712	1.319	2.6E-422	2.6E-422	34
EXACT					**1.057**	**0.781**	**1.354**			
1.2	1.3	0.6	5	500	0.921	0.710	1.300	2.3E-201	2.2E-189	238
1.2	1.3	0.6	5	1000	1.055	0.790	1.334	3.6E-407	1.1E-405	152
EXACT					**1.400**	**0.935**	**1.497**			
1.8	1.3	0.527	5	500	0.186	0.144	1.295	6.4E-195	6.4E-186	366
EXACT					**1.895**	**1.113**	**1.702**			
10.0	1.3	0.68	5	500	0.079	undef[3]	1.272	1.1E-146	1.6E-127	88
EXACT					**2.852**	**1.298**	**2.198**			

TABLE VII

Results (mean values) for queue 2 in the model with substituted queue AGG ($\#Jobs = 3$)

answers which will not conform to the current number of blocked processes. But now, we will not forget these "additional departures", which can not be realised due to a low population of blocked customers. We also do not store these additonal departures in the counter *BlockProcs* possibly resulting in zero time delays. Instead, we now count those additonal departure as "virtual arrivals" for the next Δt time interval by setting the variable *arrivals* appropriately. The modified definition of the FU *Aggregate* is shown in Fig. 8.

The results of some experiments with this redefined FU *Aggregate* are shown in Table XII.

The shown results show a similar quality as the results from our first experiments.

We assume that one reason for the bad quality of the aggregation results is given by the different views FU *IOHMM* and FU *Aggregate* do have on the current state of the overall aggregate, concerning the number of blocked processes. In the next section we are going to define a different use of IOHMMs eliminating these different views.

X. Following a new approach

In order to eliminate the different (implicitely) encoded information on the current number of blocked processes, we decide to give the IOHMM more information. The IOHMM now does not only get the information on the number of arrivals during a time interval Δt (and the corresponding number of departures), but also gets the information on the current state of the agggregate, which is (in our case) the number of blocked processes. This results in "valid answers" for the IOHMM, since invalid answers will never be encountered during the training phase. E.g., the IOHMM is trained with the input symbol (a, s) (encoded somehow) where a denotes the number of arrivals in the last observed Δt time units and s the number of currently blocked processes. The corresponding output symbol (d, s') gives the number of departures d during the last Δt time units and s' is the successor state after all currently d blocked processes have left the FU *Aggregate*. A valid "answer" of the IOHMM must obviously satisfy

$$s' = s - d \tag{3}$$

Note that $a \leq s$ and $s' \leq s$.

Since the IOHMM is trained with valid sequences (i.e. $s' \geq 0$), there is a better chance that we will "receive" only valid answers when using the IOHMM.

This design of using IOHMMs for building aggregates also reduces our problems on interpreting the IOHMM's answers. The only decision left is what to "answer" when the IOHMM is asked with a symbol $((a, s))$ not having occurred during the training phase. As before, we decided to use an identity mapping for those cases, since the envisaged application for our aggregates is for analysis of models in steady-state. I.e., whenever the IOHMM is asked with a formerly unobserved symbol (a, s) the answer will be $(a, s-a)$. Note that this is a valid answer.

The precise definition of both FUs in *ProC/B* notation is depicted in Figs. 9 and 10. First results for the (closed) tandem QN with 5 jobs are shown in

n_1	0	1	2	3
$\mu_1 = 0.1, \mu_2 = 1.3,$ **average time in states** $= 9.286$				
$\pi(n_1, n_2)$	0.00042	0.0.05462	0.07100	0.92310
time in state (n_1, n_2)	0.769	0.7143	0.7143	10.0
mean time between to consecutive visits to state $(n_1, n_2) = \frac{\text{Time in } (n_1, n_2)}{\pi(n_1, n_2)}$	1830.77	130.77	10.06	10.83
$\mu_1 = 0.9, \mu_2 = 1.3,$ **average time in states** $= 0.759$				
$\pi(n_1, n_2)$	0.133	0.191	0.277	0.399
time in state (n_1, n_2)	0.769	0.455	0.455	1.111
mean time between to consecutive visits to state $(n_1, n_2) = \frac{\text{Time in } (n_1, n_2)}{\pi(n_1, n_2)}$	5.803	2.374	1.644	2.782
$\mu_1 = 1.2, \mu_2 = 1.3,$ **average time in states** $= 0.603$				
$\pi(n_1, n_2)$	0.221	0.239	0.259	0.281
time in state (n_1, n_2)	0.769	0.4	0.4	0.833
mean time between to consecutive visits to state $(n_1, n_2) = \frac{\text{Time in } (n_1, n_2)}{\pi(n_1, n_2)}$	3.483	1.672	1.543	2.968
$\mu_1 = 1.8, \mu_2 = 1.3,$ **average time in states** $= 0.527$				
$\pi(n_1, n_2)$	0.382	0.276	0.199	0.144
time in state (n_1, n_2)	0.769	0.323	0.323	0.556
mean time between to consecutive visits to state $(n_1, n_2) = \frac{\text{Time in } (n_1, n_2)}{\pi(n_1, n_2)}$	2.016	1.170	1.621	3.865
$\mu_1 = 10.0, \mu_2 = 1.3,$ **average time in states** $= 0.681$				
$\pi(n_1, n_2)$	0.870	0.113	0.0147	0.00191
time in state (n_1, n_2)	0.769	0.0885	0.0885	0.1
mean time between to consecutive visits to state $(n_1, n_2) = \frac{\text{Time in } (n_1, n_2)}{\pi(n_1, n_2)}$	0.884	0.782	6.017	52.303

TABLE VIII
FURTHER RESULTS OF THE QN OF FIG. 4 FOR $\#Jobs = 3, n_2 = 3 - n_1$

Table XIII[4]. The training data was taken from observations of a simulation of a single M/M/1 queue with service rate $\mu = 2.0$ now embedded in an open(!) environment with a Poisson arrival stream with rate $\lambda = 1.0$. Again, the training data comprises only situations observed after 5000 time units with the additional restriction that observations are only allowed to start after the queue has been emptied.[5]

The next table shows some information on the training data compared to theoretical values. We compared the number of departures for a given Δt value with the Poisson probability distribution for rate $\lambda = 1.0$. Note that the departure process of a M/M/1 queue (where jobs arrive according to a Poisson distribution) is again Poisson with the same rate. Ta-

[4] "#applications of identity mapping" is the number of outputs(!) "WARNING: data.a exceeds number of symbols..." for applications of "Identity_Mapping" in FU *IOHMM*

[5] So we run the simulation for 5000 time units and afterwards checked (at departures) whether the queue has been emptied. If so, we started tracing arrivals/departures.

Δt	N	T	population	throughput	sojourn time	sum of all differences *(departures - 1) - max_stopped_procs*
EXACT			**2.267**	**1.038**	**2.185**	
0.8	6	1002	1.82	0.93	1.96	685
0.8	15	202	1.76	0.93	1.90	626
0.8	15	1002	1.72	0.91	1.90	642
0.3	6	102	1.31	0.80	1.64	250
0.3	6	1003	1.72	0.93	1.85	619
0.3	10	90	1.98	0.94	2.10	2697
* 0.3	10	1003	2.21	1.04	2.13	1388
0.3	15	102	1.73	0.90	1.93	1483
* 0.3	15	1003	2.20	1.03	2.12	1361
* 0.3	25	1003	2.20	1.04	2.13	1387
0.1	6	1004	1.87	0.94	1.99	930
0.1	15	1004	1.85	0.94	1.97	913
0.1	15	3003	1.79	0.93	1.91	838
0.1	30	277	1.51	0.86	1.76	455

TABLE IX
ADDITIONAL RESULTS (LAST COLUMN) FOR QUEUE 2 IN THE MODEL WITH SUBSTITUTED QUEUE AGG ($\#Jobs = 5, \mu_1 = 1.2, \mu_2 = 1.3$)

ble XIV compares the training data with theoretical values indicating the quality of the training data used for the experiments.

Table XIII shows that the new approach (taking the state information into account for training) leads to satisfactory results for some parameters. Especially the results for $\Delta t \in [0.2, 0.3]$ for large observation sequences ($T > 700$) give a good approximation for the performance measures of queue 2.

XI. CONCLUSIONS

This report describes first results of various experiments where input-output hidden Markov chains have been used to build aggregates for performance models. It provides a snapshot of activities for building aggregates in logistics networks. The positive results shown table XIII give some hope to determine accurate aggregates also in non-product form environments and for more complex sub-models. Next experiments will have to consider possible dependencies of model parameters and corresponding training data as well as parameters for the IOHMM, like Δt and T.

REFERENCES

[1] J. Banks (eds.). *Handbook of simulation. Principles, Methodology, Advances, Applications, and Practice.* John Wiley & Sons, 1998.

[2] F. Bause, H. Beilner. Intrinsic Problems in Simulation of Logistic Networks. *11th European Simulation Symposium and Exhibition* (ESS'99), Simulation in Industry, Erlangen, October 26-28, pp. 193-198, 1999.

[3] F. Bause, H. Beilner. Private Communications. March 2003.

[4] F. Bause, H. Beilner, M. Fischer, P. Kemper, M. Völker. The Proc/B Toolset for the Modelling and Analysis of Process Chains. in: T. Field, P.G. Harrison, J. Bradley, U. Harder (eds): *Computer Performance Evaluation, Modelling Techniques and Tools*, Lecture Notes in Computer Science, No 2324, Springer, pp. 51-70, 2002.

[5] Y. Bengio. Markovian Models for Sequential Data. Neural Computing Surveys 2, pp. 129-162, 1999. see also http://www.icsi.berkeley.edu/ jagota/NCS

[6] G. Bolch, S. Greiner, H. de Meer, K.S. Trivedi. Queueing Networks and Markov Chains. *J. Wiley & Sons*, 1998.

[7] P. Buchholz, H. Stahl. Construction Techniques for Aggregates. Project Report R5.4-4 of the Integrated Modelling Support Environment (IMSE) project, 1990.

[8] K. Kant. Introduction to computer system performance evaluation. *Mc Graw Hill*, (1992).

[9] B. Knab. Erweiterungen von Hidden-Markov-Modellen zur Analyse ökonomischer Zeitreihen. Dissertation, Mathematisch-Naturwissenschaftliche Fakultät Köln, 2000.

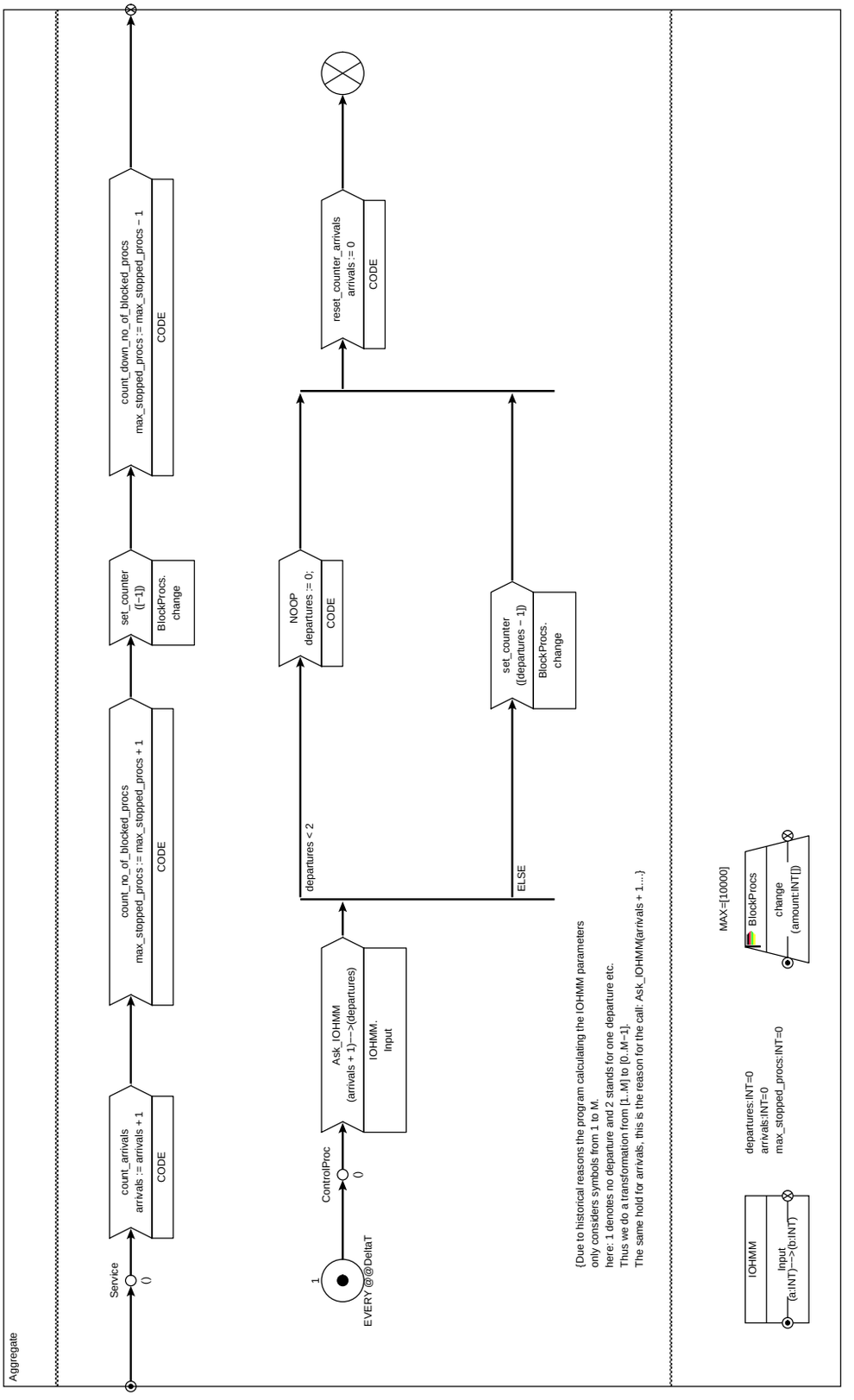


Fig. 6. Modified Version of Template IOHMM Aggregate

μ_1	μ_2	Δt	N	T	population	throughput	sojourn time	sum of all differences *(departures - 1) - max_stopped_procs*	sum of usage of identity mapping *(= sum of (inp. symbol - M))*
1.2	1.3	0.8	6	1002	2.417	1.014	2.384	2476	4
1.2	1.3	0.8	15	202	2.264	0.994	2.278	2099	20
1.2	1.3	0.8	15	1002	2.269	0.994	2.282	2138	3
1.2	1.3	0.3	6	102	1.524	0.848	1.797	779	82
1.2	1.3	0.3	6	1003	2.224	0.994	2.238	2173	3
1.2	1.3	0.3	10	90	3.782	1.150	3.290	7868	202
1.2	1.3	** 0.3	10	1003	3.783	1.183	3.195	7477	21
1.2	1.3	0.3	15	102	2.778	1.033	2.689	4172	144
1.2	1.3	** 0.3	15	1003	3.715	1.179	3.149	7254	2
1.2	1.3	** 0.3	25	1003	3.727	1.175	3.169	7231	14
1.2	1.3	0.1	6	1004	2.840	1.052	2.700	3837	19
1.2	1.3	0.1	15	1004	2.786	1.057	2.636	3836	19
1.2	1.3	0.1	15	3003	2.552	1.020	2.503	2984	21
1.2	1.3	0.1	30	277	1.878	0.911	2.060	1513	411
1.2	1.3	0.537	10	502	2.852	1.088	2.620	4148	13
1.2	1.3	0.537	10	1003	2.384	1.002	2.378	2543	5
1.2	1.3	0.4	10	502	2.404	1.004	2.394	2711	23
1.2	1.3	0.4	10	1002	2.517	1.021	2.465	3016	1
5 JOBS; EXACT					**2.267**	**1.038**	**2.185**		
0.9	1.3	0.7	10	502	1.979	0.893	2.217	1951	11
5 JOBS; EXACT					**1.508**	**0.851**	**1.773**		
1.8	1.3	0.48	10	503	3.293	1.130	2.912	5477	7
5 JOBS; EXACT					**3.392**	**1.217**	**2.787**		
0.1	1.3	9.0	5	252	0.088	0.109	0.811	0	0
0.1	1.3	9.0	5	502	0.502	0.334	1.504	0	0
3 JOBS; EXACT					**0.083**	**0.099**	**0.832**		
0.9	1.3	0.75	5	502	1.227	0.803	1.527	1860	40
0.9	1.3	0.75	5	1005	1.213	0.797	1.523	1717	38
3 JOBS; EXACT					**1.057**	**0.781**	**1.354**		
1.2	1.3	0.6	5	502	1.141	0.775	1.473	1499	24
1.2	1.3	0.6	5	1002	1.478	0.906	1.632	2933	43
3 JOBS; EXACT					**1.400**	**0.935**	**1.497**		
1.8	1.3	0.527	5	502	0.414	0.244	1.697	791	13
3 JOBS; EXACT					**1.895**	**1.113**	**1.702**		

TABLE X

RESULTS (MEAN VALUES) FOR QUEUE 2 USING MODIFIED AGGREGATE FOR QUEUE *AGG*

[10] A. Law, W. Kelton. *Simulation modeling and analysis.* 3rd ed., McGraw Hill, 2000.

[11] Z. Michalewicz, D.B. Fogel. How to solve it: Modern Heuristics. Springer, 2000.

[12] M.C. Ring. MAPM, A Portable Arbitrary Precision Math Library in C. C/C++ Users Journal, Vol. 19, No 11, November 2001. see also http://www.tc.umn.edu/ ringx004/mapm-main.html

[13] W. J. Stewart. *Introduction to the numerical solution of Markov chains.* Princeton University Press, 1994.

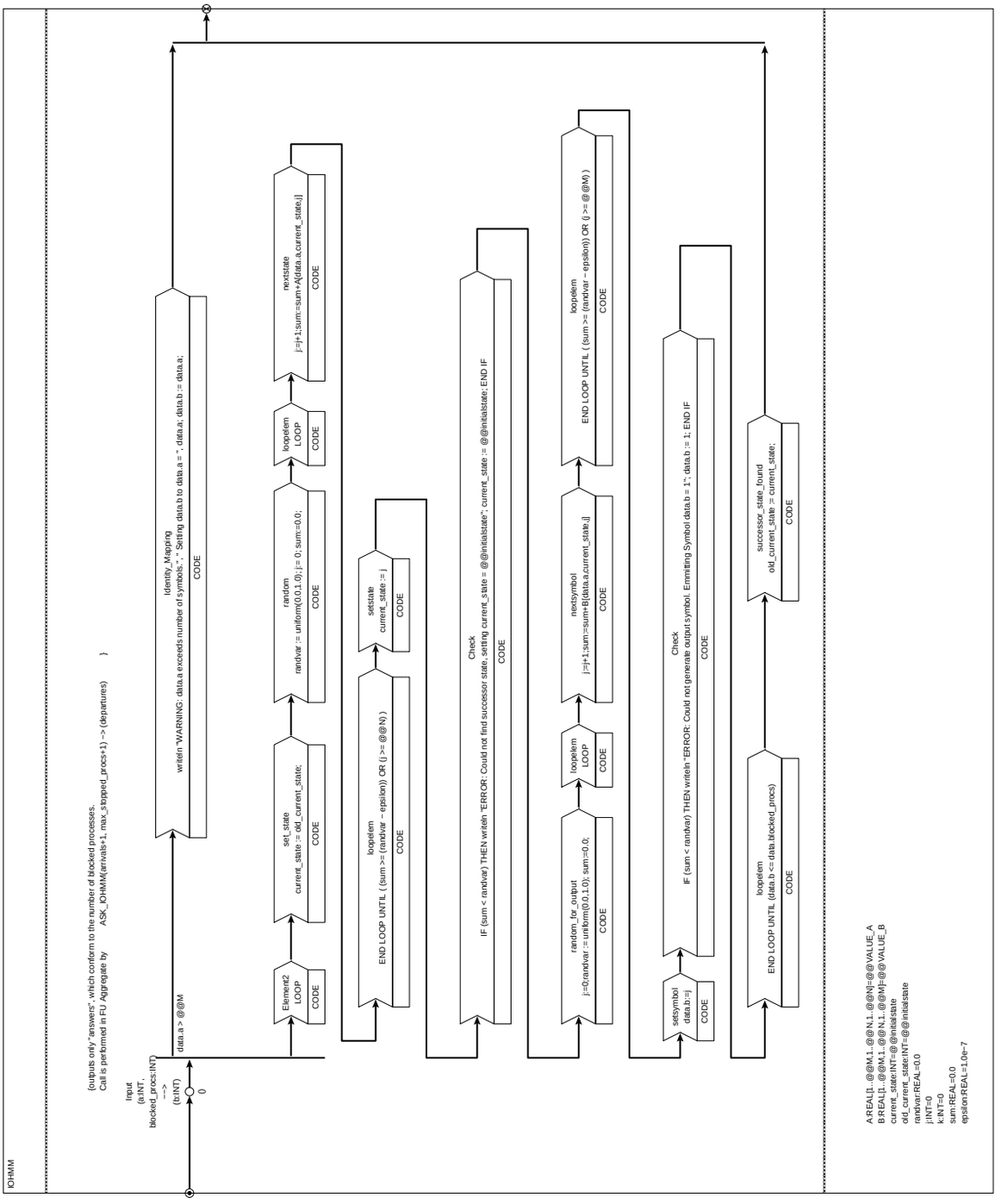


Fig. 7. Modified Version of Template IOHMM

μ_1	μ_2	Δt	N	T	population	throughput	sojourn time	sum of all differences *(departures - 1) - max_stopped_procs*	sum of usage of identity mapping *(= sum of (inp. symbol - M))*
1.2	1.3	0.8	6	1002	1.721	0.918	1.875	0	0
1.2	1.3	0.8	15	202	1.697	0.923	1.839	0	12
1.2	1.3	0.8	15	1002	1.649	0.905	1.822	0	0
1.2	1.3	0.3	6	102	1.302	0.816	1.595	0	62
1.2	1.3	0.3	6	1003	1.676	0.924	1.814	0	1
1.2	1.3	0.3	10	90	1.957	0.946	2.070	0	124
1.2	1.3	** 0.3	10	1003	2.156	1.029	2.096	0	7
1.2	1.3	0.3	15	102	1.663	0.899	1.849	0	69
1.2	1.3	** 0.3	15	1003	2.147	1.027	2.091	0	1
1.2	1.3	** 0.3	25	1003	2.130	1.028	2.072	0	9
1.2	1.3	0.1	15	3003	1.774	0.930	1.906	0	18
1.2	1.3	0.1	30	277	1.488	0.862	1.728	0	352
1.2	1.3	0.537	10	502	1.833	0.944	1.942	0	1
1.2	1.3	0.537	10	1003	1.667	0.925	1.802	0	1
1.2	1.3	0.4	10	502	1.659	0.910	1.824	0	11
1.2	1.3	0.4	10	1002	1.714	0.938	1.827	0	2
5 JOBS; EXACT					**2.267**	**1.038**	**2.185**		
1.8	1.3	0.48	10	503	2.063	0.980	2.106	0	3
5 JOBS; EXACT					**3.392**	**1.217**	**2.787**		
0.1	1.3	9.0	5	252	0.088	0.109	0.811	0	0
0.1	1.3	9.0	5	502	0.502	0.334	1.504	0	0
3 JOBS; EXACT					**0.083**	**0.099**	**0.832**		
0.9	1.3	0.75	5	502	0.916	0.721	1.271	0	0
0.9	1.3	0.75	5	1005	0.880	0.697	1.263	0	0
3 JOBS; EXACT					**1.057**	**0.781**	**1.354**		
1.2	1.3	0.6	5	502	0.873	0.694	1.260	0	0
1.2	1.3	0.6	5	1002	1.009	0.769	1.312	0	0
3 JOBS; EXACT					**1.400**	**0.935**	**1.497**		
1.8	1.3	0.527	5	502	0.219	0.173	1.263	0	0
3 JOBS; EXACT					**1.895**	**1.113**	**1.702**		

TABLE XI

RESULTS (MEAN VALUES) FOR QUEUE 2 USING MODIFIED DEFINITION OF FU IOHMM FOR QUEUE *AGG*

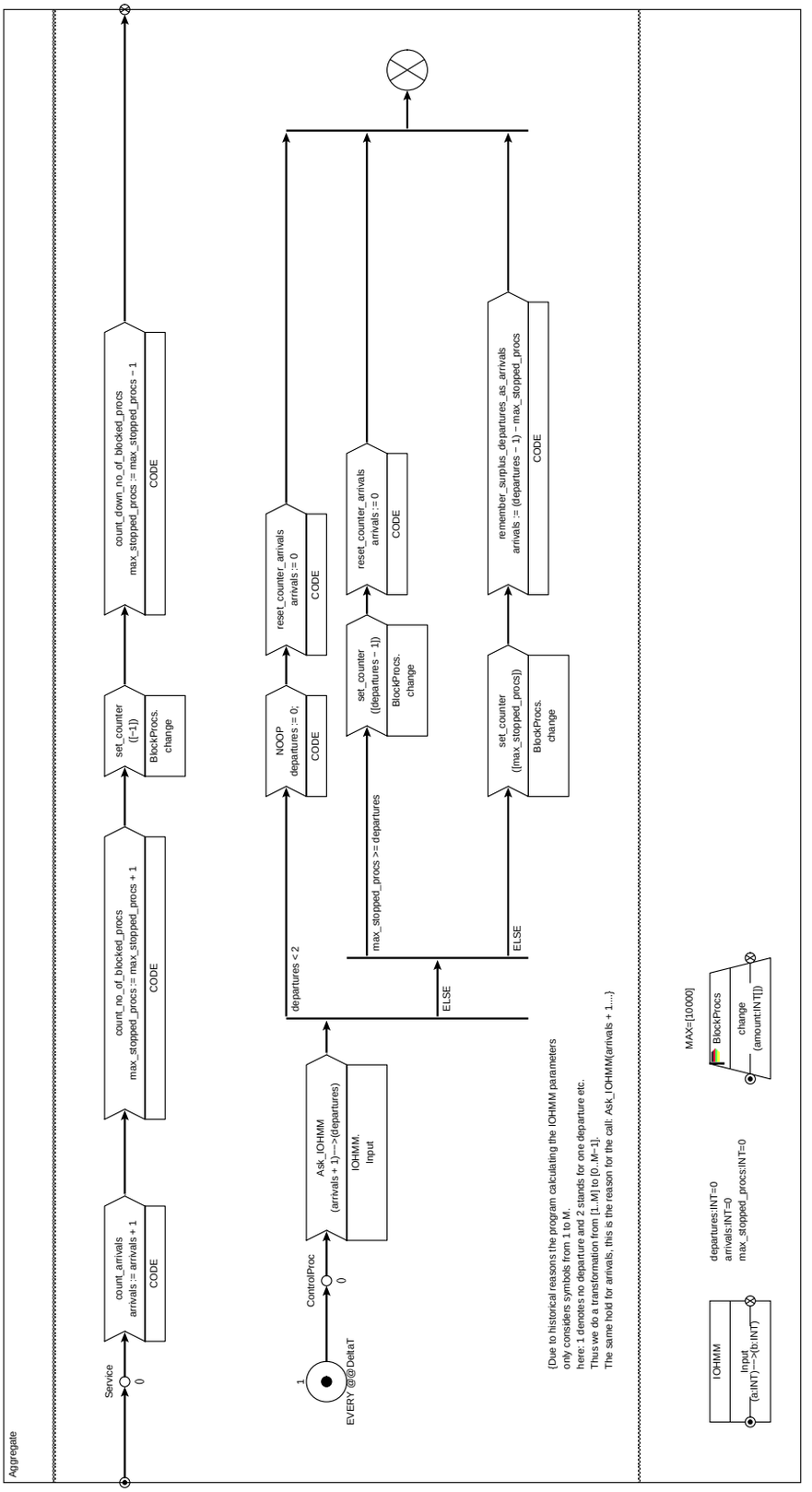


Fig. 8. Further Redefinition of Template IOHMM Aggregate

μ_1	μ_2	Δt	N	T	population	throughput	sojourn time	sum of all differences *(departures - 1) - max_stopped_procs*	sum of usage of identity mapping *(= sum of (inp. symbol - M))*
1.2	1.3	0.8	6	1002	1.838	0.934	1.968	883	16
1.2	1.3	0.8	15	202	1.848	0.938	1.970	791	54
1.2	1.3	0.8	15	1002	1.782	0.922	1.933	750	6
1.2	1.3	0.3	6	102	1.309	0.820	1.597	293	92
1.2	1.3	0.3	6	1003	1.721	0.930	1.849	651	1
1.2	1.3	0.3	10	90	2.010	0.956	2.103	1946	295
1.2	1.3	** 0.3	10	1003	2.214	1.042	2.125	1548	52
1.2	1.3	0.3	15	102	1.727	0.898	1.921	1060	184
1.2	1.3	** 0.3	15	1003	2.196	1.035	2.121	1361	1
1.2	1.3	** 0.3	25	1003	2.204	1.035	2.129	1387	8
1.2	1.3	0.1	15	3003	1.791	0.934	1.917	870	21
1.2	1.3	0.1	30	277	1.542	0.868	1.777	512	435
1.2	1.3	0.537	10	502	1.847	0.958	1.928	1090	23
1.2	1.3	0.537	10	1003	1.781	0.936	1.902	844	21
1.2	1.3	0.4	10	502	1.734	0.924	1.876	784	36
1.2	1.3	0.4	10	1002	1.748	0.941	1.858	756	4
5 JOBS; EXACT					**2.267**	**1.038**	**2.185**		
1.8	1.3	0.48	10	503	2.155	0.992	2.174	1545	45
5 JOBS; EXACT					**3.392**	**1.217**	**2.787**		
0.1	1.3	9.0	5	252	0.088	0.109	0.811	0	0
0.1	1.3	9.0	5	502	0.502	0.334	1.504	0	0
3 JOBS; EXACT					**0.083**	**0.099**	**0.832**		
0.9	1.3	0.75	5	502	1.006	0.757	1.329	1214	166
0.9	1.3	0.75	5	1005	0.971	0.734	1.324	1037	161
3 JOBS; EXACT					**1.057**	**0.781**	**1.354**		
1.2	1.3	0.6	5	502	0.924	0.722	1.281	820	60
1.2	1.3	0.6	5	1002	1.074	0.801	1.340	1283	122
3 JOBS; EXACT					**1.400**	**0.935**	**1.497**		
1.8	1.3	0.527	5	502	0.209	0.161	1.302	259	20 (bad confidence interval of ≈ 20%)
3 JOBS; EXACT					**1.895**	**1.113**	**1.702**		

TABLE XII

RESULTS (MEAN VALUES) FOR QUEUE 2 USING REDEFINED FU *Aggregate* AND FIRST VERSION OF FU *IOHMM* FOR QUEUE *AGG*

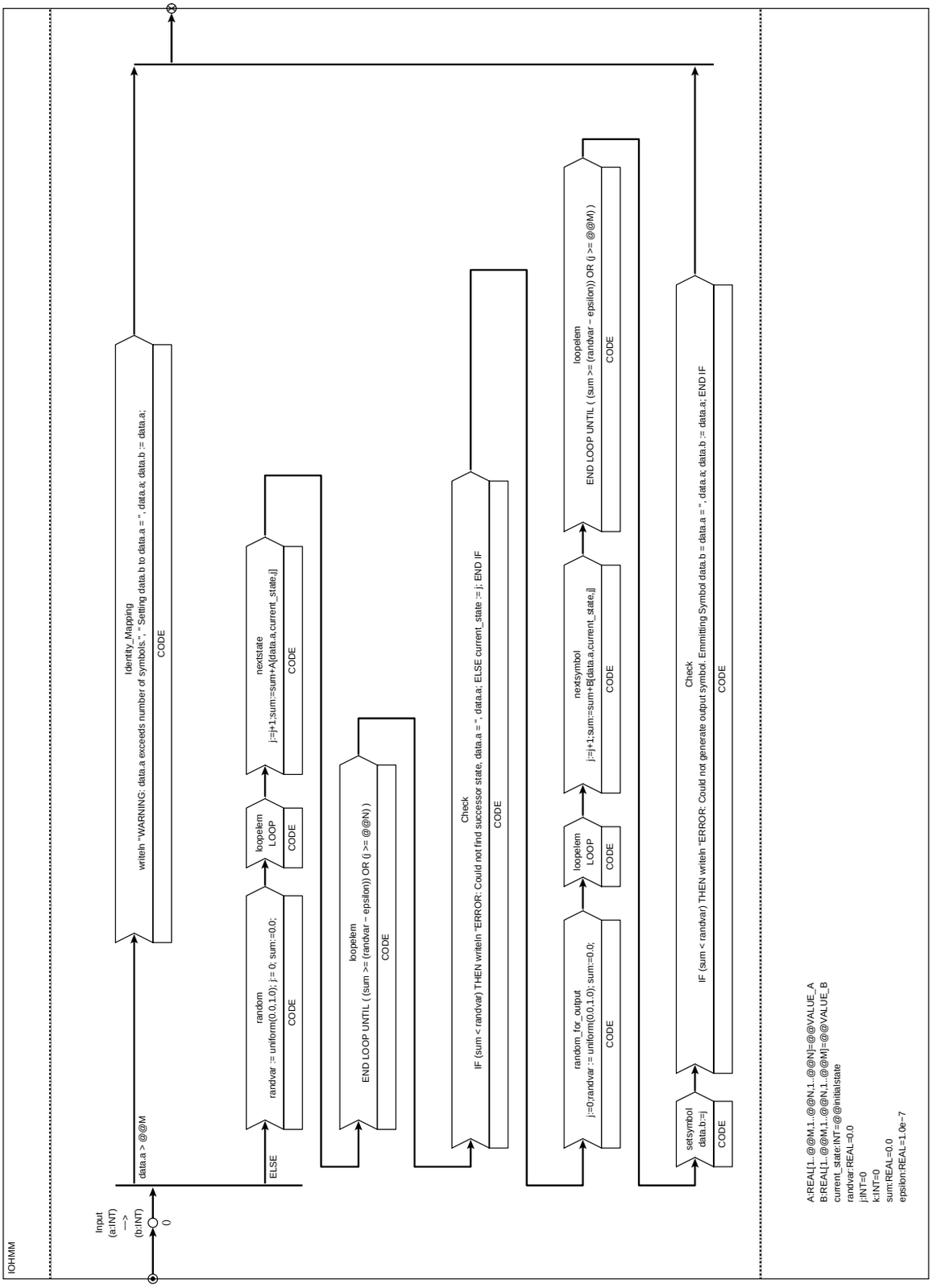


Fig. 9. Definition of Template IOHMM

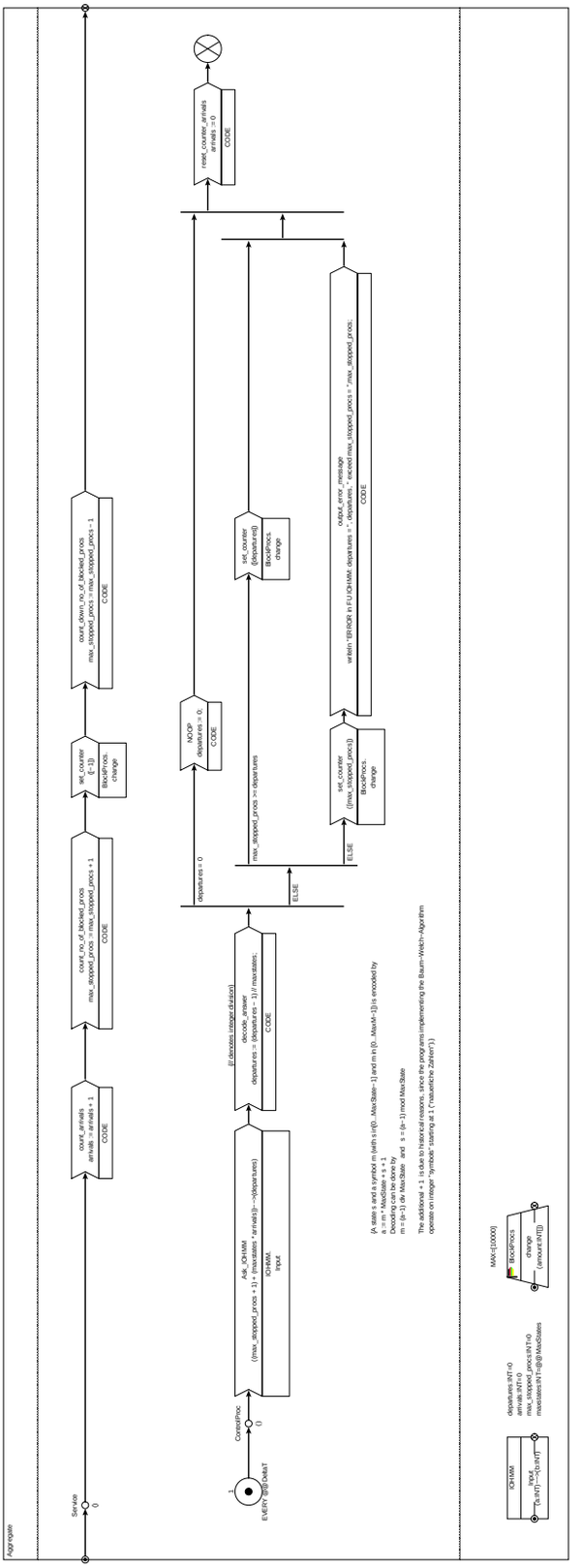


Fig. 10. Corresponding Definition of Template IOHMM Aggregate encoding arrivals and the current number of blocked processes into one input symbol for the IOHMM

μ_1	μ_2	Δt	N	T	population	throughput	sojourn time	#applications of identity mapping
2.0	1.3	0.1	10	1504	3.388	1.247	2.717	28
2.0	1.3	0.2	10	752	3.473	1.281	2.709	95
2.0	1.3	0.3	10	502	3.311	1.266	2.616	88
2.0	1.3	0.3	15	502	3.317	1.247	2.661	80
2.0	1.3	0.3	20	502	3.247	1.231	2.638	86
2.0	1.3	0.8	10	181	2.954	1.228	2.405	14
2.0	1.3	0.8	15	181	2.935	1.210	2.426	15
2.0	1.3	0.8	20	181	2.940	1.201	2.448	19
2.0	1.3	2.0	10	76	2.534	1.117	2.268	0
2.0	1.3	3.0	10	51	2.029	0.983	2.064	0
2.0	1.3	3.0	15	51	2.372	1.043	2.275	0
2.0	1.3	3.0	20	51	2.132	1.011	2.107	0
2.0	1.3	0.2	10	1514	3.694	1.273	2.902	100
2.0	1.3	0.3	3	1503	3.442	1.237	2.783	52 (bad optimisation)
2.0	1.3	0.3	6	1503	3.533	1.256	2.813	73
2.0	1.3	0.3	6	755	3.520	1.268	2.776	59
2.0	1.3	0.3	10	755	3.543	1.283	2.762	74
5 JOBS; EXACT					**3.6322**	**1.2429**	**2.9224**	

TABLE XIII

Results (mean values) for queue 2 using IOHMMs trained with input symbols (a, s) taking the current state into account

μ_1	μ_2	Δt	T	$P[0]$	$P[1]$	$P[2]$	$P[3]$	$P[4]$	$P[5]$	$P[6]$	$P[7]$	$P[8]$	$P[9]$	$P[10]$
2.0	1.3	0.1	1504	1366	129	9								
		0.1	theo	1361	136	7								
2.0	1.3	0.2	752	615	127	10								
		0.2	theo	616	123	12	1							
2.0	1.3	0.3	502	370	119	11	2							
		0.3	theo	372	112	17	2							
2.0	1.3	0.3	755	566	166	20	3							
		0.3	theo	559	168	25	3							
2.0	1.3	0.8	181	80	71	23	7							
		0.8	theo	81	65	26	7	1						
2.0	1.3	2.0	76	9	19	23	17	6	2					
		2.0	theo	10	21	21	14	7	3	1				
2.0	1.3	3.0	51	2	8	13	8	9	10	1				
		3.0	theo	3	8	11	11	9	5	3	1			

TABLE XIV

Distribution of departures: used training data vs. theoretical values.
$P[i] := P[i$ departures in interval $\Delta t]$; empty entries in columns $P[i]$ denote 0