SCHOOL OF OPERATIONS RESEARCH
AND INDUSTRIAL ENGINEERING
COLLEGE OF ENGINEERING
ITHACA, NEW YORK 14853

TECHNICAL REPORT NO. 933

September 1990

HEURISTICS FOR KANBAN ALLOCATION IN A MULTI-STAGE
STOCHASTIC ENVIRONMENT

By
Sridhar R. Tayur
Cornell University
Heuristics for kanban allocation in a multi-stage stochastic environment

Sridhar R. Tayur
Cornell University

Abstract

In this paper we develop heuristics for allocation of kanbans to cells in a balanced line. Our approach is to complement simulation with analytical tools. An interesting property of our heuristics is that the distribution of the processing times on these machines is not used in the determination of a good allocation. In fact, we do not attempt to approximate the mean throughput for each candidate allocation as part of the heuristic. Two surrogate objectives are developed which are to be maximized. The computation of the objectives requires only the knowledge of the structure of the line, namely, the number of machines and cards in each cell. These objectives accurately identify good allocations. The mean throughput for the recommended allocation is obtained via simulation. Using this approach we are able to solve large scale problems in a greater generality than previously possible.
1 Introduction

In this paper we consider the problem of allocating a given total number of kanbans to cells so as to maximize the mean throughput rate, for the case of identical machines. The model for the line and the structural properties of the line are described in [Tay 90], [Tayur 90]. We observed in the example of section 2.3 in [Tay 90] that in a general environment we apparently cannot discard any of the undominated allocations in the search for an optimal solution to this problem. The cells are predetermined in the sense that we know how many machines are there in each cell. We first consider the case of one machine per cell, and then generalize to the case where any cell can have an arbitrary finite number of machines. An interesting property of our heuristics is that the distribution of the processing times on these machines is not used in the determination of a good allocation. In fact, we do not attempt to approximate the mean throughput as part of the heuristic. Two surrogate objectives are developed which are to be maximized. These surrogate objectives yield allocations that are good for a variety of processing time distributions. The computation of the objectives requires only the knowledge of the structure of the line, namely, the number of machines and cards in each cell and not the processing time distributions of the machines.

Exact solution even for an environment with all machines having an exponential distribution seems computationally intractable for large problems. For machines that have a processing time distribution other than exponential, an exact analytic solution even for small problems may not be possible. Thus, it appears that the best method of arriving at a good allocation will probably be in two steps: for every candidate allocation calculate its measure of goodness, and then pick the allocation with the highest measure. In the past [Mitra 88] [Dele 89] [Zip 89], an estimate of the mean throughput obtained from analytical approximations was used as a measure of goodness. This severely restricts the range of processing times that can be analyzed, and takes significant computing time and space. We avoid this problem by developing surrogate measures for mean throughput (that do not depend on the distributions of the processing times, but on the number of cells and the number of kanbans in each cell) that can be computed very easily. Thus, our heuristics have the same second step as before, but the first step is extremely quick and holds in good generality. When the best
allocation has been determined using the heuristics, the mean throughput corresponding to this allocation is obtained by simulation.

In section 2.1, we consider the problem of allocating a fixed total number of kanbans among \( N \) cells, each cell containing one machine, and all of the machines being identical in the sense of having the same processing time distribution. Two heuristics for surrogate measures of throughput are developed in this section. The simulation results to verify the performance of the heuristics are tabulated in section 3.

The two heuristics did extremely well for the single machine per cell case, and led to the use of the same approach for the case where cells can consist of any number of machines. Two heuristics in the same spirit are developed in section 2.2 for this environment, and extensive simulation (shown in section 3) indicates that these heuristics do well here too. To our knowledge, for multiple machines per cell, no other computationally feasible method for the allocation of kanbans to cells has been proposed thus far.

We will use \( N/(M_1 \ldots M_N)/(C_1 \ldots C_N) \) to denote a kanban line with \( N \) cells, \( M_i \) machines in cell \( i \), and \( C_i \) kanbans in cell \( i, i = 1 \ldots N \). By allocation we mean the vector \( (C_1 \ldots C_N) \), and by partition we mean \( (M_1 \ldots M_N) \). The set \( \{ N/(M_1 \ldots M_N)/(C_1 \ldots C_N): \sum_{i=1}^{N} M_i = M, M_i \geq 1, \sum_{i=1}^{N} C_i = C, C_i \geq 1, N \leq M \} \) are all the configurations for a line with \( M \) machines and \( C \) kanbans.

2 Kanban allocation heuristics

2.1 One machine per cell

We are interested in distributing a fixed total number of kanbans among \( N \) cells, each cell containing one machine, and all the machines being identical in the sense of having the same processing time distribution. We first develop two heuristic measures of goodness for machines with exponential processing times whose performances are verified by simulation. We then provide an intuitive explanation to as to why these heuristics can be expected to work well. Based on the intuition and extensive computational results, we conclude that the same heuristics are applicable to the case with identical general machines.
The motivation for the first heuristic comes from the following example and from theorem 1 below. Without loss of generality, we assume the mean processing time to be 1.

**Example 1** Consider a two machine line with identical exponential machines. Let the total number of kanbans be \( C \), with \( C_1 \) in cell 1, and \( C_2 \) in cell 2. This can easily be modeled as a birth-death process with a total of \( C+1 \) states, namely \( C_1 \ldots + 1,0,-1,\ldots - C_2 \), each state being equally likely. Each state represents the difference between the number of cards in the output-hopper of cell 1 and on the bulletin board of the second cell. Note that these two numbers cannot be both positive at the same time. Adding an extra card to either cell has exactly the same effect on the size of the state-space and the throughput. The size of the state-space increases, and the throughput of the line increases as the probability of being in state \( C_1 \) decreases. Compare this to corollary 6 of section 3.2 in [Tay 90].

Thus, we are interested in the size of the state space of the markov chain that each kanban allocation to a given set of machines creates. The size of the state-space can be easily calculated by the following recursion. \( Y_j \) is the total number of states in a line that consists only of machines \( j \) through \( N \) (both inclusive) and \( X_j \) is the total number of states for the same line except that we allow kanbans to wait at bulletin board of cell \( j \). \( Y_1 \) gives the size of the state-space.

\[
\begin{align*}
X_N &= C_N + 1 \\
Y_N &= 1 \\
X_{n-1} &= (C_{n-1} + 1)X_n + \frac{C_{n-1}(C_{n-1}+1)}{2}Y_n \\
Y_{n-1} &= X_n + C_{n-1}Y_n
\end{align*}
\]

where \( N \) is the total number of machines, and \( (C_1 \ldots C_N) \) is the kanban allocation. A pleasantly surprising result is the following (see [Tay 90] for the definition of dominance):

**Theorem 1** If allocation \( C_A \) dominates allocation \( C_B \) then the state space generated by \( C_A \) is larger than that generated by \( C_B \) when the machines have exponential processing times.
Proof We can write the recursions above in a matrix product form, as follows:

\[
\begin{pmatrix}
X_1 \\
Y_1
\end{pmatrix} = \prod_{i=1}^{N} \begin{pmatrix}
C_i + 1 & \frac{C_i(C_i+1)}{C_i} \\
1 & 0
\end{pmatrix}
\begin{pmatrix}
1 \\
0
\end{pmatrix}
\]

One can easily verify the claim by first comparing an allocation \((C_1 \ldots C_N)\) with \((C_1, C_2, \ldots, C_{N-1} + C_N - 1, 1)\), and then comparing \((1, C_{N-1} + C_N - 1, \ldots, C_2, C_1)\) with \((1, C_{N-1} + C_N - 1, \ldots, C_2 + C_1 - 1, 1)\) (in both cases one needs to only multiply the last two matrices and the vector). Note that an allocation and another allocation that is in its reverse order have both the same size of the state space and the same mean throughput (recall the reversibility result of section 2.1 in [Tay 90]). This completes the proof that undominatable allocations have a larger state-space than the allocations they dominate. Other cases are also as easily verified.

Q.E.D.

A similar derivation yields an O(N) recursion for the state space for the case when all the machines have a processing time distribution of erlang with mean 1 and parameter k. The result is:

\[
\begin{pmatrix}
X_1^k \\
Y_1^k
\end{pmatrix} = \prod_{i=1}^{N} \begin{pmatrix}
kC_i + 1 & \frac{kC_i(C_i-1)}{k} + C_i \\
k & k(C_i - 1) + 1
\end{pmatrix}
\begin{pmatrix}
1 \\
0
\end{pmatrix}
\]

The size of the state-space is now given by \(Y_1^k\). A proof similar to theorem 1 yields the following result.

**Theorem 2** If allocation \(C_A\) dominates allocation \(C_B\) then the state space generated by \(C_A\) is larger than that generated by \(C_B\) when the machines have erlang processing times.

The following two corollaries are immediate (compare to corollaries in [Tay 90]).

**Corollary 1** All feasible allocations in a two-cell system with the same fixed number of cards yield the same state-space.

**Corollary 2** In a three cell system with a total of \(C+2\) cards, the allocation that yields the largest state-space is \((1, C, 1)\).
The strong relationship between dominance and state-space is the motivation for our heuristics. The first heuristic is to allocate kanbans so as to maximize the (size of the) state space.

We showed in [Tay 90] that the optimal allocation for a two cell or a three cell line is independent of the processing time distribution on the machines. The above heuristic conforms to the result. In a N cell line with identical machines, we expect (intuitively) that the optimal allocation is independent of the processing time distribution of the machines. That the above heuristic supports the intuition is shown by the next result. For the definition of majorization and schur-concavity, see [Marsh 79].

**Theorem 3** In a $4/(1,1,1,1)/(1,C_2, C_3, 1)$ with erlang machines with $C_2 + C_3 = C$, the state space is maximized by $(C_2^*, C_3^*)$ the vector that is majorized by every other vector $(C_2, C_3)$.

**Proof** The state space is schur-concave in $(C_2, C_3)$. This is easily verified by noting that the state space is symmetric in $C_2$ and $C_3$, and that $\frac{dY^k}{dc_2} \geq \frac{dY^k}{dc_3}$ whenever $C_2 \leq C_3$.

Q.E.D.

**Example 2** In a 4 cell line with erlang machines, with 10 cards to allocate, the state-space is maximized with the allocation $(1, 4, 4, 1)$. This is independent of the distribution.

A more general result is that in a N cell system, the state space is schur concave in $C_i$ and $C_{N-i+1}$, given that $C_j = C_{N_j+1}$ for all $j \neq i$. This result has the implication that the allocation that maximizes the state space can be obtained with less number of computations than exhaustive search. As an example, in a 5 cell line with C cards to allocate, we need to do $O(C)$ computations.

For the second heuristic, define

$$\text{pulse} = \frac{\sum_{s \in S} \mu(s)}{NS}$$

where $s$ is a state, $S$ is the state-space, and $N$ is the total number of cells. $\mu(s)$ is the number of machines that are working in state $s$, and takes on values between 1 and $N$. 

6
Example 3 Consider a two-machine line with exponential machines as before. Then \( \text{pulse} = \frac{1+2(C-1)+1}{2(C+1)} \frac{C}{C+1} \) = mean throughput rate.

The pulse can be quickly calculated by the following recursion. We describe the derivation of the recursion briefly for machines with exponential processing times. We work backwards from cell \( N \) to cell 1. At every stage \( n-1 \), the idea is to add a 1 if a particular state has machine \( n \) working or add a 0 if machine \( n \) is idle. When considering stage \( n-1 \), the superscript \( L \) is used when the assumption that there is no bulletin board for cell \( n-1 \) is in effect, and no superscript is used when we are assuming that there is a bulletin board in cell \( n-1 \). Note that this is only for the derivation; all cells except cell 1 have a bulletin board in the real system. We count the number of states that do not have the machine in working state, and the number of states that have machine \( n \) working. In both of the counting processes, it is important to distinguish cases that do or do not contain a part in the output hopper of cell \( n-1 \). As an example, consider the recursion for \( P_{n-1}^L \). First, let all \( C_{n-1} \) cards be at machine \( n-1 \). In this case, to \( P_n \) we add \( X_n \) (the number of states that have machine \( n-1 \) working and the output hopper of cell \( n-1 \) is empty). Next let \( a \) cards be in front of machine \( n-1 \), with \( 0 < a < C_{n-1} \). Then for each \( a \) we have, \( Y_n \) states that have machine \( n-1 \) working and output hopper of cell \( n-1 \) non-empty. So we add \( Y_n \) to \( P_n^L \). \( (C_{n-1} - 1) \) times. Finally, for the case when all the cards of cell \( n-1 \) are in the output hopper of cell \( n-1 \), we add a 0 (as machine \( n-1 \) is not working) to \( P_n^L \). This completes the step. \( P_{n-1} \) is computed in a similar way. Dividing \( P_1^L \) by the total number of states gives the average number of machines working in any state. Dividing this average by \( N \) gives the pulse.

\[
P_N = \frac{P_1^L}{N}.
\]

\[
P_{n-1}^L = P_n + X_n + C_{n-1}P_n^L + (C_{n-1} - 1)Y_n.
\]

\[
P_{n-1} = (1 + C_{n-1})P_n + C_{n-1}X_n + \frac{(C_{n-1} - 1)}{2}P_n^L + (\frac{C_{n-1}^2}{2} - 1)Y_n.
\]

We want \( P_1^L/(NY_1) \), which is the pulse for a \( N \) cell system with identical exponential machines with the allocation \( (C_1 \ldots C_N) \). The heuristic is to allocate kanbans so as to maximize the pulse. A similar derivation can be made for the case when the machines have erlang processing time distributions.
The pulse, like the size of the state-space, is only an indicator of a good allocation. *The throughput should still be estimated from simulation.* The advantage of our heuristics is that they are good indicators of good allocations, and they can be computed very quickly, in $O(N)$ time, unlike simulation or other analytic methods that take a significant amount of time per allocation tested. Thus, in our method various candidate allocations can be tried using the state-space or the pulse heuristics as measures of goodness, and the final few can be checked by simulation. Extensive simulation tests using a variety of processing time distributions, and on various different lengths of the line show that these heuristics work very well in identifying a good allocation. Some of the results are tabulated in section 3. Briefly, for cases when the ratio of total number of kanbans to number of machines is small ($\leq 1.5$) the state space is an excellent indicator, while for larger ratios of total number of kanbans to number of cells ($\geq 2$) the pulse is an excellent indictor. For the ratios between 1.5 and 2.0, both the heuristics coincide to pick a good allocation.

The following intuitive explanation for the success of the heuristics seems reasonable (this is not a proof). Observe that the pulse will yield the mean throughput rate if the transition matrix of the markov chain created by the allocation is doubly stochastic and irreducible. This is because the probability of being in any state would be the same. Thus, the success of the pulse heuristic depends on the closeness of the true transition matrix to a doubly stochastic irreducible matrix. The closeness can be measured in many ways: we compare the column sum vector of the transition matrix, obtained by adding all the elements in a given column, to a vector of ones of the same length. The transition matrices that arise due to kanban allocations in our setting are not doubly stochastic in general, but have the following special properties, which can easily be verified by noting that only certain kinds of transitions can take place. For sake of simplicity, we assume that the allocation is $(C, \ldots C)$.

- All non-zero elements in a row are equal.
- All non-zero elements are of the form $1/n$ for some $n \in 1 \ldots N$.
- The matrix has a lot of zero elements.
- The column sum of any column is at least $\frac{1}{N}$.  

8
• The column sum cannot exceed \( N(1 + \frac{1}{2} + \ldots + \frac{1}{N}) \) and this bound is independent of the allocation.

• The size of the matrix is \( O(C^N) \).

Thus, as \( C \) increases, the matrix becomes more sparse. Let \( Q_C \) be the transition matrix, and \( D_C \) be a doubly stochastic matrix of the same size as \( Q_C \). Let \( \pi_C \) and \( \pi_0 \) be the steady state probabilities due to \( Q_C \) and \( D_C \) respectively. Then

\[
\pi_0 - \pi_C = \lim_{n \to \infty} (\pi_0 D^n_C - \pi_0 Q^n_C) \\
= \frac{1}{S_C} \varepsilon \lim_{n \to \infty} (D^n_C - Q^n_C) \\
= \frac{1}{S_C} \lim_{n \to \infty} d^n_C
\]

where \( \varepsilon \) is a vector of ones, \( S_C \) is the size of the state-space, and \( d^n_C \) is the column deviation vector of \( Q^n_C \), defined as the difference of the column sum vector and \( \varepsilon \). Computational experience shows that \( d^n_C \) is of the same order as \( d^0_C \), and so

\[
\lim_{\varepsilon \to \infty} \| (\pi_C - \pi_0) \| = \lim_{\varepsilon \to \infty} \lim_{n \to \infty} \frac{1}{S_C} d^n_C \\
\approx \lim_{\varepsilon \to \infty} \frac{1}{S_C} d^1_C \\
\to 0
\]

where \( \| a \| \) is the maximum of the absolute values of the elements of the vector \( a \). At low values of \( C \), the maximum of the absolute value of \( a \) is far from 0. The deviation of the column sum vector from \( \varepsilon \) becomes less and less as the number of kanbans is increased, so the pulse can be expected to be a better indicator at high values of \( C \). While \( C \) is small, the size of the state-space makes a weaker assumption on the uniformity of probabilities among the states (assumes only that probability of being in a 'good' state is proportional to the number of good states), and so can be expected to do better. This explains the behaviour of the heuristics.

The theorem below justifies the use of the pulse as a surrogate measure for throughput.
Theorem 4 For a fixed $N$, if the allocation is $(C, \ldots, C)$, the pulse yields the throughput as $C \to \infty$.

Proof We need to show that $\frac{P}{N_1} \to 1$ as $C \to \infty$.

\[
\begin{pmatrix}
X \\
Y \\
P^L \\
N_{n+1}
\end{pmatrix} = \begin{pmatrix}
C + 1 & \frac{C(C+1)}{2} & 0 & 0 \\
1 & \frac{C}{2} & 0 & 0 \\
C & \frac{C(C-1)}{2} & 1 + C & \frac{C(C+1)}{2} \\
1 & C - 1 & 1 & \frac{C}{2}
\end{pmatrix}^{n-1} \begin{pmatrix}
C + 1 \\
1 \\
C \\
1
\end{pmatrix}
\]

is the complete recursion ($n \geq 1$) for an $N$ cell line with the allocation $(C, \ldots, C, C)$. Note that both $Y_{N-n+1}$ and $P^L_{N-n+1}$ are polynomials in $C$ for each $n$. For asymptotical analysis, it is therefore sufficient to keep track of the coefficients of the highest order term at each step.

Let $a_n$, $b_n$, $c_n$ and $d_n$ be the coefficients of the highest order term of $X_{N-n+1}$, $Y_{N-n+1}$, $P^L_{N-n+1}$ and $P_{N-n+1}$. Also, let $p_n$, $q_n$, $r_n$ and $s_n$ be the powers of the leading term of $X_{N-n+1}$, $Y_{N-n+1}$, $P^L_{N-n+1}$ and $P_{N-n+1}$. Then, $a_1 = b_1 = c_1 = d_1 = 1$. Also, $p_1 = 1$, $q_1 = 0$, $r_1 = 0$, and $s_1 = 1$. We will show that the ratio $c_n/b_n = n$ for all $n = 1 \ldots N$, that $p_n = s_n = n$, and that $q_n = r_n = n - 1$. The proof is by induction. The crux of the inductive hypothesis is $c_j = j b_j$ for all $j = 1 \ldots n$, and $p_j = j$, $q_j = j - 1$, $r_j = j - 1$ and $s_j = j$ for all $j = 1 \ldots n$. The base case is for $n = 1$, and the hypothesis is trivially true.

We have the following recursions which are obtained by matching the coefficients of the highest order term going from $n$ to $n+1$.

\[
a_{n+1} = a_n + \frac{b_n}{2} \\
b_{n+1} = a_n + b_n \\
c_{n+1} = a_n + b_n + c_n + d_n \\
d_{n+1} = a_n + \frac{b_n}{2} + \frac{c_n}{2} + d_n
\]

The inductive step is

\[
c_{n+1} = a_n + b_n + c_n + d_n \\
= b_{n+1} + c_n + d_n \\
= b_{n+1} + n b_n + d_n
\]
But, expanding $d_n$ and repeatedly applying the inductive hypothesis and the recursions,

$$
\begin{align*}
    d_n &= a_{n-1} + \frac{nb_{n-1}}{2} + d_{n-1} \\
    &= a_{n-1} + n \frac{b_{n-1}}{2} + \left( a_{n-2} + \frac{(n-1)b_{n-2}}{2} + d_{n-2} \right) \\
    & \vdots \\
    &= a_{n-1} + \ldots + a_1 + \frac{1}{2}(nb_{n-1} + \ldots + 2b_1) + 1 \\
    &= (a_{n-2} + \frac{b_{n-2}}{2}) + \ldots + (a_1 + \frac{b_1}{2}) + a_1 + \frac{1}{2}(nb_{n-1} + \ldots + 2b_1) + 1 \\
    &= a_{n-2} + \ldots + a_1 + a_1 + \frac{1}{2}(nb_{n-1} + nb_{n-2} + (n-1)b_{n-3} + \ldots + 3b_1) + 1 \\
    & \vdots \\
    &= \frac{n}{2}(b_{n-1} + \ldots + b_1) + n
\end{align*}
$$

Now, using the recursions for $b_{n+1}$ and $a_n$, we have

$$
\begin{align*}
    b_{n+1} &= a_n + b_n \\
    &= b_n + \left( a_{n-1} + \frac{b_{n-1}}{2} \right) \\
    & \vdots \\
    &= b_n + \frac{1}{2}(b_{n-1} + \ldots + b_1) + 1
\end{align*}
$$

and so, $nb_{n+1} - nb_n = n + \frac{n}{2}(b_{n-1} + \ldots + b_1)$. Thus, $d_n = nb_{n+1} - nb_n$. It now follows that $c_{n+1} = (n + 1)b_{n+1}$. Clearly, the power of the leading term increases by one in every step and this verifies that $p_{n+1} = n + 1$, $q_{n+1} = n$, $r_{n+1} = n$ and $s_{n+1} = n + 1$. This completes the inductive step.

Q.E.D.

The intuitive explanation of the success of the pulse and the state-space heristics (in the case of exponential processing time distributions) encourages us to deal with more general lines. A line with identical machines with erlang processing times will have, if the markov chain transition probability
matrix were to be written down using the method of phases, a similar structure to the one discussed above, and a lot more sparsity. This then implies, using our intuitive argument, that the state-space and the pulse measures should work as well, if not better, in the case of erlang machines. Rather than deal with the state-space and pulse of the markov chain for erlangs, we simply use the same recursions that were developed for machines with exponential processing times and check (via simulation) whether they are good measures for the lines with erlang machines. That they are indeed good is shown in the next section. As any processing time distribution can be well approximated by a weighted erlang distribution, the recursions for exponentials can be applied to any set of identical machines with good results.

2.2 Lines with more than one machine per cell

The success of the pulse and the state-space heuristics in the single machine per cell case encourages us to test their performances in a line where a cell may contain more than one machine. The state-space of the markov chain for this kind of lines grows very quickly with the number of machines in a given cell. The approximate method of [Mitra 88] for exponential machines is computationally infeasible even for relatively small problems; a cell with 8 machines and 14 kanbans has over 100000 states! Furthermore, their assumption that an isolated cell perceives demands and supply from neighboring cells as a poisson process can be seriously wrong, even though all the machines have exponential processing times, as the neighboring cells may have more than one machine.

To our knowledge, for multiple machines/cell, no other method for the allocation of kanbans to cells has been proposed for machines with exponential processing times. Our heuristics below are easy to code and do not grow enormously with the problem size, and almost always pick the best allocation. In the cases where the machine processing times are not exponentially distributed, there is no computationally (or, in many cases analytically) feasible method to compute the throughput rate and simulation is essential. Also, most lines that are operating well do not have machines with exponential processing times.

For a line with $N$ cells, and $M_i$ (exponential) machines in each cell (total $M$) with an allocation $(C_1,\ldots,C_N)$ for $i = 1\ldots N$, the size of the
state space can be quickly computed by the following recursion. Let $Y_i(l)$ stand for the number of states that have exactly $l$ machines busy, in a line that consists only of cells $i$ through $N$ inclusive. As before, we need $X_i(l)$, which is the number of states that have exactly $l$ machines busy in a line that consists of cells $i$ through $N$ and in which the cards can stay in the bulletin board of cell $i$. Thus, $Y_i(l)$ gives the number of states that have exactly $l$ machines busy, for $l = 0 \ldots M$. The derivation of the recursion is simply a counting process, and the counting is done from cell $N$ backwards to cell 1. The basic technique is similar to the one explained in section 2.1 for the derivation of the pulse for the case of one machine per cell. Here, in the formula for $Y_n(l)$ and $X_n(l)$ below, $l_1$ stands for the number of machines that are working in cell $n$, and $C_n - a$ is the number of cards that are either in the output hopper of cell $n$ or in the bulletin board of cell $n$. The case of $l_1 = 0$ is treated as a separate term for convinience.

The end-conditions are

$$Y_N(0) = 0$$

$$Y_N(l) = \binom{M_N}{l} \binom{C_N - 1}{C_N - l}$$

$$X_N(0) = 1$$

$$X_N(l) = \sum_{a=l}^{C_N} \binom{M_N}{l} \binom{a - 1}{a - l}$$

where the terms in the parenthesis are binomial coefficients. For $n = N - 1, \ldots 1$, we have

$$Y_n(0) = 0$$

$$Y_n(l) = \sum_{1 \leq l_1 \leq l} \sum_{a=l_1}^{C_{n-1}} \binom{M_n}{l_1} \binom{a - 1}{a - l_1} Y_{n+1}(l - l_1)$$

$$+ \binom{M_n}{l_1} \binom{C_n - 1}{C_n - l_1} X_{n+1}(l - l_1) + Y_{n+1}(l)$$
\[ X_n(0) = 1 \]

\[ X_n(l) = \sum_{1 \leq i \leq l} \sum_{a = l_1}^{C_n} \left[ \binom{M_n}{l_1} \left( \frac{a - 1}{a - l_1} \right) ((C_n - a)Y_{n+1}(l - l_1) + X_{n+1}(l - l_1)) \right] + C_nY_{n+1}(l) + X_{n+1}(l) \]

The complexity of this recursion is \( O(NM^2C^2) \). As in the case of one machine per column, the transition matrix of the Markov chain has a special structure, and a similar intuitive explanation motivates the use of the pulse and the state-space heuristics. As a counterpart to theorem 1 of section 2.1, we have the following result.

**Theorem 5** (a) For a \( 2/(1,M)/(C_1,C_2) \) line with a total of \( C \) cards, the allocation that yields the highest state-space is \( (1, C-1) \).

(b) For a \( N/(1, M_2 \ldots M_{N-1}, 1)/(C_1 \ldots C_N) \) line, with a total of \( C \) cards, the allocation that yields the highest state-space has exactly one card in each of the end cells.

**Proof** The proofs are similar to that of theorem 1.

Q.E.D.

Similar results hold when the machines have Erlang processing time distributions. Before we show the computational results (section 3.3), we have another result. For a single cell in isolation, with \( M \) machines and \( C \) cards, the pulse is equal to the mean throughput. This is because the transition matrix is doubly stochastic. Combining the results above and the intuitive explanations of section 2.1, we can expect the following: (a) in a line with a large number of cards, the pulse heuristic will do well in identifying the best allocation while the state-space will perform well when the number of cards are low, and (b) in a line where most cells contain more than one machine, the pulse heuristic will be a good indicator.
3 Performance of State-Space and Pulse heuristics

3.1 Introduction

An extensive simulation experiment was conducted to determine the following:

1. the performance of heuristics as the length of the line varies,

2. the performance of heuristics as the variability of the processing times increases,

3. the general shape of the allocation that the heuristics determine, and

4. the sensitivity of throughput to the competing allocations.

Each simulation was run for 20000 completions of parts in cell \( N \), and data collection began after the first 1000 outputs. We set up the simulations so that the mean processing times are 1 on all machines. A batched mean estimator was used for estimating the throughput, and the confidence interval was \( \pm 0.0015 \). The time required to compute the size of the state-space and the pulse for each allocation is negligible. Simulation times were linear in the number of cells, and did not depend on number of kanbans to be allocated (note the obvious advantage over analytic methods aiming to estimate throughput for exponential processing times etc.). On the average, a 12 cell line took 1.5 minutes, a 16 cell line 2.0 minutes, 20 cell line 2.5 minutes, and a 30 cell line took 3.45 minutes on a SPARC station 1. The processing times used varied from hyper-exponential distributions to erlang distributions, the length of the lines varied from 12 to 30 cells, and the number of kanbans to be allocated varied from 16 to 61. The experiments were setup so that we can vary the length of the line, and for each fixed length we can see the performance of our heuristics at different throughput levels (by increasing the total number of kanbans to be allocated), and for fixed total number of kanbans we can study the effect of different processing time distributions.

In each table the allocation, the state space, the pulse and the throughput due to the appropriate processing time distribution are provided. Due
to space constraints, the pulse is truncated to four decimals, and an approximate value of the state space is given if it gets too large.

In each table, only the closest of the competing allocations are shown. Many other allocations were tested. For the first few cases in small lines, almost all undominated allocations were tested to gain an intuition about the general shape of good allocations, and to verify that our heuristics will discard the obviously bad allocations. As an example, in a four cell line with a total of 10 cards, the allocation (1,7,1,1) cannot possibly compete to be the best. However, we tested it to make sure that our heuristics will punish such an allocation. For longer lines, based on the intuition gained from the small lines, only the competing allocations were tested.

Between the two heuristics there is not much difference, and they usually coincide in the choice of the best allocation. When they don't, we use the state-space for the cases when the ratio of the total number of kanbans to the number of cells is small ($\leq 1.5$), and when the ratio is large ($\geq 2.0$) we use the pulse. In the range 1.5 - 2.0, both the heuristics coincided and picked good allocations. Once the allocation with the highest state-space or pulse was obtained, simulation was performed for all allocations that are its neighbors and on all the competing (in the sense of state-space or pulse) allocations. By neighbor we mean an allocation that is obtained from moving a card out of a cell to an adjacent cell. As an example, for a four cell line, a neighbor of the allocation (1,4,4,1) is the allocation (1,5,3,1). The tables show only the closest of the allocations that were tested. The testing of all the neighbors was done to determine if the allocations selected by our heuristics were a local maximum.

### 3.2 One machine per cell

Our computational experience for the heuristics on lines with machines that have an exponential processing time distribution is shown in the tables 1 and 2. Only the competing allocations are shown in the tables. The general shape of good allocations may seem a little strange as they don't conform to the bowl shape that one might expect. The dynamic behaviour of the cards, mentioned in section 2, causes good allocations to have peaks at cell 2 and cell $N-1$. In a 4 cell line, all but two cards should be assigned to the middle two cells. By symmetry one can anticipate that the two middle cells should differ by at most one (compare theorem 3). In a 5 cell
Table 1: State-Space and Pulse in a line with exponential machines.

<table>
<thead>
<tr>
<th>allocation</th>
<th>states</th>
<th>pulse</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,2,1,2,1</td>
<td>153</td>
<td>0.6209</td>
<td>0.5366</td>
</tr>
<tr>
<td>1,2,2,1,1</td>
<td>151</td>
<td>0.6146</td>
<td>0.5261</td>
</tr>
<tr>
<td>1,2,2,2,1</td>
<td>241</td>
<td>0.6564</td>
<td>0.6219</td>
</tr>
<tr>
<td>1,2,3,1,1</td>
<td>222</td>
<td>0.6397</td>
<td>0.6103</td>
</tr>
<tr>
<td>1,3,2,1,1</td>
<td>219</td>
<td>0.6411</td>
<td>0.5919</td>
</tr>
<tr>
<td>1,3,2,2,1</td>
<td>348</td>
<td>0.6828</td>
<td>0.6465</td>
</tr>
<tr>
<td>1,2,3,2,1</td>
<td>345</td>
<td>0.6788</td>
<td>0.6467</td>
</tr>
<tr>
<td>1,3,2,3,1</td>
<td>501</td>
<td>0.7090</td>
<td>0.6776</td>
</tr>
<tr>
<td>1,2,3,3,1</td>
<td>489</td>
<td>0.7035</td>
<td>0.6708</td>
</tr>
<tr>
<td>1,4,3,3,1</td>
<td>922</td>
<td>0.7451</td>
<td>0.7166</td>
</tr>
<tr>
<td>1,3,4,3,1</td>
<td>906</td>
<td>0.7415</td>
<td>0.7132</td>
</tr>
<tr>
<td>1,4,3,4,1</td>
<td>1228</td>
<td>0.7622</td>
<td>0.7362</td>
</tr>
<tr>
<td>1,4,4,3,1</td>
<td>1196</td>
<td>0.7578</td>
<td>0.7313</td>
</tr>
<tr>
<td>1,5,4,4,1</td>
<td>2000</td>
<td>0.7863</td>
<td>0.7609</td>
</tr>
<tr>
<td>1,4,5,4,1</td>
<td>1960</td>
<td>0.7833</td>
<td>0.7593</td>
</tr>
<tr>
<td>1,7,5,8,1</td>
<td>7906</td>
<td>0.8455</td>
<td>0.8272</td>
</tr>
<tr>
<td>1,8,4,8,1</td>
<td>7836</td>
<td>0.8454</td>
<td>0.8267</td>
</tr>
<tr>
<td>1,7,6,7,1</td>
<td>7903</td>
<td>0.8450</td>
<td>0.8265</td>
</tr>
<tr>
<td>1,3,2,2,2,1</td>
<td>6441</td>
<td>0.6701</td>
<td>0.6240</td>
</tr>
<tr>
<td>1,2,2,3,2,2,1</td>
<td>6366</td>
<td>0.6664</td>
<td>0.6248</td>
</tr>
</tbody>
</table>
Table 2: State-Space and Pulse in a 4 cell line with exponential machines.

<table>
<thead>
<tr>
<th>Allocation</th>
<th>States</th>
<th>Pulse</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,3,2,1</td>
<td>81</td>
<td>0.6944</td>
<td>0.6708</td>
</tr>
<tr>
<td>1,4,1,1</td>
<td>72</td>
<td>0.6701</td>
<td>0.6300</td>
</tr>
<tr>
<td>1,3,3,1</td>
<td>115</td>
<td>0.7261</td>
<td>0.7056</td>
</tr>
<tr>
<td>1,4,2,1</td>
<td>110</td>
<td>0.7182</td>
<td>0.6924</td>
</tr>
<tr>
<td>1,4,4,1</td>
<td>204</td>
<td>0.7696</td>
<td>0.7529</td>
</tr>
<tr>
<td>1,5,3,1</td>
<td>198</td>
<td>0.7652</td>
<td>0.7452</td>
</tr>
<tr>
<td>1,6,5,1</td>
<td>405</td>
<td>0.8130</td>
<td>0.7996</td>
</tr>
<tr>
<td>1,7,4,1</td>
<td>390</td>
<td>0.8083</td>
<td>0.7927</td>
</tr>
</tbody>
</table>

line the middle cell can have a much lower number of cards than its two neighbours ((1,7,5,8,1) is better than (1,7,6,7,1)). Symmetric allocation with the maximum number of cards in the middle need not be the best (for example, (1,4,3,3,1) is better than (1,3,4,3,1)). A greedy algorithm to allocate cards one at a time will not work, as, for example, adding a card to the allocation (1,1,2,1,1) is inferior to (1,2,1,2,1). Thus, it is the end effects and the dynamic character of the cards that cause good allocations in small lines to look very different from a bowl shape. Put differently, one should look for the bowl phenomenon not in the cards \( C_k \), but rather in the sums \( C_k + C_{k+1} \) and \( C_{k-1} + C_k + C_{k+1} \).

Similar results were found with a greater number of kanbans to be allocated, i.e., at higher throughputs. Although a large number of simulations were performed, we show here only a sample that are typical of the results obtained.

Table 3 shows the throughput as obtained from simulation when the processing times on the machines have an erlang distribution. In every case, both the pulse and the state-space heuristics coincided in identifying the best allocation. As before, only a fraction of our tests are reported, and only competing allocations are shown. Further testing indicated that it is very difficult to find cases when the heuristics are off the mark! Thus, the heuristics performed better than expected. They performed better than in
Table 3: Throughput in lines with erlang machines.

<table>
<thead>
<tr>
<th>allocation</th>
<th>states</th>
<th>pulse</th>
<th>Erlang 2</th>
<th>Erlang 3</th>
<th>Erlang 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,3,3,1</td>
<td>115</td>
<td>0.7261</td>
<td>0.7988</td>
<td>0.8459</td>
<td>0.8941</td>
</tr>
<tr>
<td>1,4,2,1</td>
<td>110</td>
<td>0.7182</td>
<td>0.7848</td>
<td>0.8317</td>
<td>0.8834</td>
</tr>
<tr>
<td>1,1,2,1,1</td>
<td>94</td>
<td>0.5723</td>
<td>0.6286</td>
<td>0.6803</td>
<td>0.7413</td>
</tr>
<tr>
<td>1,2,1,1,1</td>
<td>92</td>
<td>0.5739</td>
<td>0.6151</td>
<td>0.6606</td>
<td>0.7261</td>
</tr>
<tr>
<td>1,2,2,1,2</td>
<td>153</td>
<td>0.6209</td>
<td>0.6828</td>
<td>0.7400</td>
<td>0.8403</td>
</tr>
<tr>
<td>1,2,2,1,1</td>
<td>151</td>
<td>0.6146</td>
<td>0.6681</td>
<td>0.7200</td>
<td>0.7925</td>
</tr>
<tr>
<td>1,3,2,2,1</td>
<td>348</td>
<td>0.6828</td>
<td>0.7492</td>
<td>0.8036</td>
<td>0.8616</td>
</tr>
<tr>
<td>1,2,3,2,1</td>
<td>345</td>
<td>0.6788</td>
<td>0.7483</td>
<td>0.8014</td>
<td>0.8583</td>
</tr>
<tr>
<td>1,3,2,2,3,1</td>
<td>2160</td>
<td>0.6972</td>
<td>0.7670</td>
<td>0.8204</td>
<td>0.8758</td>
</tr>
<tr>
<td>1,2,3,2,2,1</td>
<td>2094</td>
<td>0.6537</td>
<td>0.7569</td>
<td>0.8099</td>
<td>0.8649</td>
</tr>
<tr>
<td>1,3,2,2,2,2,1</td>
<td>6441</td>
<td>0.6701</td>
<td>0.7348</td>
<td>0.7932</td>
<td>0.8528</td>
</tr>
<tr>
<td>1,2,2,3,2,2,1</td>
<td>6366</td>
<td>0.6664</td>
<td>0.7347</td>
<td>0.7914</td>
<td>0.8517</td>
</tr>
<tr>
<td>1,4,3,3,3,3,1</td>
<td>33178</td>
<td>0.7418</td>
<td>0.8078</td>
<td>0.8567</td>
<td>0.9033</td>
</tr>
<tr>
<td>1,3,3,4,3,3,1</td>
<td>32394</td>
<td>0.7383</td>
<td>0.8056</td>
<td>0.8540</td>
<td>0.9010</td>
</tr>
<tr>
<td>1,2,2,1,2,1</td>
<td>660</td>
<td>0.6247</td>
<td>0.6869</td>
<td>0.7453</td>
<td>0.8116</td>
</tr>
<tr>
<td>1,2,2,2,1,1</td>
<td>650</td>
<td>0.6187</td>
<td>0.6706</td>
<td>0.7246</td>
<td>0.7819</td>
</tr>
</tbody>
</table>
the case of identical exponential machines.

A 12 cell line with three levels of total number of kanbans to be allocated is considered first with machines that have exponential processing times (tables 4-7). Next, a 16 machine line is tested at three levels of total number of kanbans to be allocated (tables 8-10). This is followed by a 20 machine line with two levels of total number of kanbans to be allocated (table 11). The effect of processing time distribution is studied in a 30 cell line (table 12), followed by the study of the effect of machine breakdowns (table 13). A brief description of each table is given below followed by a summary of general findings.

Tables 4 - 6 show the results of our heuristics on a 12 cell line with 16, 17 and 26 cards respectively when the processing times are all exponentially distributed. All allocations that are the nearest neighbors to the one with the highest state-space were simulated to check whether we had a local maximum according to throughput. Note that in table 4 all but 4 cells have one card each, and the rest have two cards each. The cells that have two cards are used as labels for the allocation. Thus, \((3,5,7,9)\) means that the third, fifth, seventh and ninth cells have two cards each, and the rest of the cells have one card each. Two neighbors of \((3,5,7,9)\) are \((2,5,7,9)\) and \((4,5,7,9)\) in this notation. In tables 4 and 5, we achieved a local maximum, as in many of the cases below. Also, other allocations (in particular, ones with 3 cards in a cell) are not shown simply because they are not competitive. For the case of 26 cards, we provide two tables (6 and 7) that show all the competing allocations and the neighbors. As in the cases of 16 and 17 cards, a local maximum was obtained.

In table 8, a 16 cell line with 21 cards is considered. All the nearest neighbors of \((3,5,8,11,14)\) were checked. Again, the state-space heuristic picked the best.

In table 9, for a 26 card allocation problem with 16 cells, we see that although both the pulse and state-space heuristics coincide, they pick an inferior allocation. Checking all nearest neighbors of \((4,7,10,13)\) showed that it was a local maximum, although it was not a global maximum ((3,6,10,14) did 0.5% better).

In table 10, we show that concentrating in the middle at the expense of long segments is not good. Also, having lopsided allocations in interior cells yields low throughput. The point is that our heuristics will indicate a bad allocation. This is indeed the case in all of the previous tables too; we
Table 4: A 12/(1...1)/() line with a total of 16 cards. Cell no. indicates cells with 2 cards.

<table>
<thead>
<tr>
<th>cell no.</th>
<th>states</th>
<th>pulse</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>3,5,8,10</td>
<td>400,725</td>
<td>0.5529</td>
<td>0.4894</td>
</tr>
<tr>
<td>3,6,7,10</td>
<td>389,081</td>
<td>0.5549</td>
<td>0.4866</td>
</tr>
<tr>
<td>2,5,7,10</td>
<td>392,739</td>
<td>0.5539</td>
<td>0.4854</td>
</tr>
<tr>
<td>2,4,7,10</td>
<td>392,529</td>
<td>0.5539</td>
<td>0.4835</td>
</tr>
<tr>
<td>3,6,9,11</td>
<td>392,529</td>
<td>0.5539</td>
<td>0.4827</td>
</tr>
<tr>
<td>2,5,8,11</td>
<td>384,865</td>
<td>0.5549</td>
<td>0.4811</td>
</tr>
<tr>
<td>2,4,8,11</td>
<td>383,043</td>
<td>0.5549</td>
<td>0.4764</td>
</tr>
<tr>
<td>3,6,8,10</td>
<td>400,713</td>
<td>0.5529</td>
<td>0.4895</td>
</tr>
<tr>
<td>3,5,7,10</td>
<td>400,713</td>
<td>0.5529</td>
<td>0.4887</td>
</tr>
<tr>
<td>3,5,8,9</td>
<td>388,557</td>
<td>0.5507</td>
<td>0.4869</td>
</tr>
<tr>
<td>2,5,8,10</td>
<td>392,715</td>
<td>0.5538</td>
<td>0.4852</td>
</tr>
<tr>
<td>3,5,8,11</td>
<td>392,715</td>
<td>0.5539</td>
<td>0.4851</td>
</tr>
<tr>
<td>3,4,8,10</td>
<td>387,267</td>
<td>0.5510</td>
<td>0.4818</td>
</tr>
</tbody>
</table>

Table 5: A 12/(1...1)/() line with a total of 17 cards. Cell no. indicates cells with 2 cards. Other cells have one card each.

<table>
<thead>
<tr>
<th>cell no.</th>
<th>states</th>
<th>pulse</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>3,5,7,9,11</td>
<td>665,811</td>
<td>0.5721</td>
<td>0.5080</td>
</tr>
<tr>
<td>2,5,7,9,11</td>
<td>652,563</td>
<td>0.5731</td>
<td>0.5024</td>
</tr>
<tr>
<td>3,6,7,9,11</td>
<td>646,053</td>
<td>0.5702</td>
<td>0.5016</td>
</tr>
<tr>
<td>2,5,8,9,11</td>
<td>633,117</td>
<td>0.5712</td>
<td>0.4956</td>
</tr>
<tr>
<td>2,5,8,10,11</td>
<td>630,699</td>
<td>0.5714</td>
<td>0.4928</td>
</tr>
<tr>
<td>3,5,7,8,10</td>
<td>659,043</td>
<td>0.5692</td>
<td>0.5080</td>
</tr>
<tr>
<td>3,5,7,8,9</td>
<td>635,577</td>
<td>0.5668</td>
<td>0.5026</td>
</tr>
<tr>
<td>4,5,7,8,9</td>
<td>615,795</td>
<td>0.5646</td>
<td>0.4969</td>
</tr>
</tbody>
</table>
Table 6: A $12/(1\ldots1)/()$ line with a total of 26 cards. Cell no. indicates cells with 3 cards. The other interior cells have 2 cards.

<table>
<thead>
<tr>
<th>cell no.</th>
<th>states</th>
<th>pulse</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>2,5,8,11</td>
<td>27,837,756</td>
<td>0.6854</td>
<td>0.6270</td>
</tr>
<tr>
<td>2,5,7,10</td>
<td>27,480,708</td>
<td>0.6834</td>
<td>0.6266</td>
</tr>
<tr>
<td>2,4,7,10</td>
<td>27,488,322</td>
<td>0.6834</td>
<td>0.6265</td>
</tr>
<tr>
<td>2,4,8,11</td>
<td>27,786,888</td>
<td>0.6853</td>
<td>0.6261</td>
</tr>
<tr>
<td>3,5,8,10</td>
<td>27,131,112</td>
<td>0.6815</td>
<td>0.6261</td>
</tr>
<tr>
<td>3,6,9,11</td>
<td>27,488,322</td>
<td>0.6834</td>
<td>0.6257</td>
</tr>
<tr>
<td>3,6,7,10</td>
<td>26,772,768</td>
<td>0.6808</td>
<td>0.6250</td>
</tr>
</tbody>
</table>

Table 7: Testing nearest neighbors of (2,5,8,11).

<table>
<thead>
<tr>
<th>cell no.</th>
<th>states</th>
<th>pulse</th>
<th>throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>2,5,8,10</td>
<td>27,482,166</td>
<td>0.6834</td>
<td>0.6269</td>
</tr>
<tr>
<td>3,6,8,10</td>
<td>27,128,196</td>
<td>0.6814</td>
<td>0.6265</td>
</tr>
<tr>
<td>3,5,8,11</td>
<td>27,482,166</td>
<td>0.6834</td>
<td>0.6262</td>
</tr>
<tr>
<td>3,5,7,10</td>
<td>27,128,196</td>
<td>0.6814</td>
<td>0.6257</td>
</tr>
<tr>
<td>3,4,8,10</td>
<td>26,730,081</td>
<td>0.6808</td>
<td>0.6249</td>
</tr>
<tr>
<td>3,5,9,11</td>
<td>27,432,189</td>
<td>0.6834</td>
<td>0.6248</td>
</tr>
<tr>
<td>3,5,8,9</td>
<td>26,664,876</td>
<td>0.6805</td>
<td>0.6244</td>
</tr>
<tr>
<td>3,5,9,10</td>
<td>26,730,081</td>
<td>0.6808</td>
<td>0.6242</td>
</tr>
<tr>
<td>4,5,8,10</td>
<td>26,664,487</td>
<td>0.6805</td>
<td>0.6240</td>
</tr>
</tbody>
</table>
Table 8: A 16/(1...1)/() line with 21 cards. Cell no. indicates cells with two cards.

<table>
<thead>
<tr>
<th>cell no.</th>
<th>states</th>
<th>pulse</th>
<th>throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>3,5,8,11,14</td>
<td>32,583,330</td>
<td>0.5397</td>
<td>0.4663</td>
</tr>
<tr>
<td>3,7,9,11,14</td>
<td>32,445,792</td>
<td>0.5397</td>
<td>0.4659</td>
</tr>
<tr>
<td>4,6,8,10,12</td>
<td>32,347,908</td>
<td>0.5392</td>
<td>0.4654</td>
</tr>
<tr>
<td>2,5,8,11,13</td>
<td>31,890,054</td>
<td>0.5402</td>
<td>0.4647</td>
</tr>
<tr>
<td>3,6,9,12,15</td>
<td>31,932,020</td>
<td>0.5405</td>
<td>0.4638</td>
</tr>
<tr>
<td>2,4,7,10,13</td>
<td>31,872,810</td>
<td>0.5403</td>
<td>0.4628</td>
</tr>
<tr>
<td>2,5,7,11,14</td>
<td>31,797,798</td>
<td>0.5405</td>
<td>0.4613</td>
</tr>
</tbody>
</table>

Table 9: A 16/(1...1)/() line with 26 cards. Cell no. indicates the interior cells with one card.

<table>
<thead>
<tr>
<th>cell no.</th>
<th>states</th>
<th>pulse</th>
<th>throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>3,6,10,14</td>
<td>372,527,829</td>
<td>0.6021</td>
<td>0.5309</td>
</tr>
<tr>
<td>3,7,10,14</td>
<td>372,530,907</td>
<td>0.6021</td>
<td>0.5305</td>
</tr>
<tr>
<td>4,7,10,13</td>
<td>373,737,483</td>
<td>0.6027</td>
<td>0.5285</td>
</tr>
<tr>
<td>2,4,13,15</td>
<td>356,498,415</td>
<td>0.5971</td>
<td>0.5226</td>
</tr>
<tr>
<td>3,8,9,14</td>
<td>357,837,975</td>
<td>0.6003</td>
<td>0.5216</td>
</tr>
<tr>
<td>2,8,9,15</td>
<td>346,812,595</td>
<td>0.5957</td>
<td>0.5193</td>
</tr>
<tr>
<td>2,3,14,15</td>
<td>330,956,483</td>
<td>0.5935</td>
<td>0.5075</td>
</tr>
</tbody>
</table>
Table 10: A $16/(1\ldots1)/(\ldots)$ line with 35 cards. Cell no. indicates cells with 3 cards, * indicates cells with 4 cards, and + indicates cells with 1 card.

<table>
<thead>
<tr>
<th>cell no.</th>
<th>states, in $10^9$</th>
<th>pulse</th>
<th>throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>2,5,8,12,15</td>
<td>13.60</td>
<td>0.6802</td>
<td>0.6138</td>
</tr>
<tr>
<td>3,5,8,11,14</td>
<td>13.29</td>
<td>0.6773</td>
<td>0.6127</td>
</tr>
<tr>
<td>4,6,8,10,12</td>
<td>13.13</td>
<td>0.6768</td>
<td>0.6111</td>
</tr>
<tr>
<td>$2^<em>,3^</em>,4,13^<em>,15^</em>$</td>
<td>11.69</td>
<td>0.6756</td>
<td>0.6052</td>
</tr>
<tr>
<td>7,8,9,10,11</td>
<td>12.21</td>
<td>0.6745</td>
<td>0.6048</td>
</tr>
<tr>
<td>$2^<em>,8,15^</em>$</td>
<td>12.32</td>
<td>0.6787</td>
<td>0.6027</td>
</tr>
</tbody>
</table>

just did not show the obviously bad allocations before.

Table 11 shows allocations for a 20 cell line. The difference in throughput among competing allocations is not significant (as always). The appropriate heuristic (state-space for low number of kanbans, and pulse for large number of kanbans) picked well.

We conclude the section by showing the performance of our heuristics on lines with machines that have processing time distributions more variable than the exponential distribution (tables 12 and 13). A higher variability than an exponential distribution can happen when the machines are subject to (infrequent) long breakdowns, but operate very fast when they are up. We consider two hyper-exponential distributions, one with a mean of 0.1333 and made up of exponential(0.4) and exponential(0.04), and the other with mean 0.1 made up of exponential(0.6) and exponential(0.06).

Table 12, shows the performance of the state-space heuristic on a 30 cell line for machines with hyper-exponential, exponential and erlang processing times. The point is that our heuristics perform well for a wide variety of distributions even in long lines.

Table 13 shows a five cell line with machines that have a hyper-exponential processing time distribution. The results are similar to the exponential case except that the difference in throughput between allocations is lower near the local maximum.

To summarize briefly, the performance of our heuristics is robust with respect to the length of the line, and to the total number of kanbans to be
Table 11: A 20/(1...1)/(/) line with C cards. Cell no. indicates interior cells with one extra card compared to other interior cells.

<table>
<thead>
<tr>
<th>C = 25</th>
<th>cell no.</th>
<th>states, in $10^8$</th>
<th>pulse</th>
<th>throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4,7,11,14,17</td>
<td>15.50</td>
<td>0.5206</td>
<td>0.4437</td>
</tr>
<tr>
<td></td>
<td>4,7,10,14,18</td>
<td>15.46</td>
<td>0.5208</td>
<td>0.4428</td>
</tr>
<tr>
<td></td>
<td>3,8,11,14,18</td>
<td>15.41</td>
<td>0.5211</td>
<td>0.4413</td>
</tr>
<tr>
<td></td>
<td>4,9,11,13,17</td>
<td>15.36</td>
<td>0.5207</td>
<td>0.4410</td>
</tr>
<tr>
<td></td>
<td>3,7,11,15,19</td>
<td>15.05</td>
<td>0.5216</td>
<td>0.4376</td>
</tr>
<tr>
<td></td>
<td>2,6,11,15,19</td>
<td>14.67</td>
<td>0.5222</td>
<td>0.4335</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C = 61</th>
</tr>
</thead>
<tbody>
<tr>
<td>cell no.</td>
</tr>
<tr>
<td>2,7,10,14,19</td>
</tr>
<tr>
<td>2,8,10,13,19</td>
</tr>
<tr>
<td>3,7,10,14,18</td>
</tr>
<tr>
<td>4,7,10,14,17</td>
</tr>
<tr>
<td>3,6,10,15,18</td>
</tr>
<tr>
<td>4,8,10,13,17</td>
</tr>
<tr>
<td>5,8,10,13,16</td>
</tr>
<tr>
<td>8,9,10,11,12</td>
</tr>
</tbody>
</table>
Table 12: A \(30/(1\ldots1)/(\) line with 35 cards. Cell no. indicates cells with 2 cards.

<table>
<thead>
<tr>
<th>cell no.</th>
<th>states, in (10^{12})</th>
<th>hyper1</th>
<th>exp</th>
<th>erl(2)</th>
<th>erl(3)</th>
<th>erl(4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5,10,15,20,25</td>
<td>23.54</td>
<td>0.2238</td>
<td>0.4076</td>
<td>0.5108</td>
<td>0.5726</td>
<td>0.6149</td>
</tr>
<tr>
<td>6,11,17,22,27</td>
<td>23.53</td>
<td>0.2232</td>
<td>0.4068</td>
<td>0.5086</td>
<td>0.5708</td>
<td>0.6122</td>
</tr>
<tr>
<td>4,10,16,22,26</td>
<td>23.53</td>
<td>0.2232</td>
<td>0.4059</td>
<td>0.5084</td>
<td>0.5705</td>
<td>0.6115</td>
</tr>
<tr>
<td>2,8,13,18,24</td>
<td>22.87</td>
<td>0.2220</td>
<td>0.4051</td>
<td>0.5065</td>
<td>0.5666</td>
<td>0.6100</td>
</tr>
<tr>
<td>3,9,15,21,28</td>
<td>23.35</td>
<td>0.2211</td>
<td>0.4030</td>
<td>0.5045</td>
<td>0.5654</td>
<td>0.6074</td>
</tr>
<tr>
<td>2,3,4,28,29</td>
<td>19.37</td>
<td>0.2116</td>
<td>0.3830</td>
<td>0.4796</td>
<td>0.5373</td>
<td>0.5783</td>
</tr>
<tr>
<td>2,6,15,25,29</td>
<td>22.20</td>
<td>0.2182</td>
<td>0.3964</td>
<td>0.4961</td>
<td>0.5556</td>
<td>0.5967</td>
</tr>
<tr>
<td>13,14,15,16,17</td>
<td>19.64</td>
<td>0.2207</td>
<td>0.3944</td>
<td>0.4919</td>
<td>0.5505</td>
<td>0.5923</td>
</tr>
</tbody>
</table>

allocated for a given line. The general shape of the allocations depends on the number of cards that are in excess to the number of interior cells. In general, it is usually best to give 1 card each to the end cells, to give \(\lfloor \frac{2}{N} \rfloor\) to the interior cells, and then to spread the remaining kanbans uniformly along the line so that the length of the segments between these cells are approximately equal. This strategy is suggested as the starting point for the heuristics. Concentrating the excess cards in the center cells is not a good allocation scheme in long lines. This shape remains valid for a wide of distributions, over different lengths of the line, and for any number of kanbans to be allocated.

It was observed that the state-space was a better measure when the ratio of number of kanbans to the number of cells was low \((\leq 1.5)\), and the pulse was a better measure when the ratio was high \((\geq 2)\). In the range 1.5 to 2.0, both the heuristics coincided. This is in agreement with the intuition described in section 2.1.

The difference in throughputs for competing allocations is low, and differ by less than 2%. The state-space and the pulse heuristics usually pick a local minimum; when they don't, they are off by less than 0.5%. In rare cases, it is possible that, although our heuristics select a local maximum, it is not a global maximum (difference is less than 0.5%). Out of a total of
Table 13: A $5/(1\ldots1)/(\ldots)$ line.

<table>
<thead>
<tr>
<th>allocation</th>
<th>states</th>
<th>pulse</th>
<th>hyper1</th>
<th>hyper2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,2,1,2,1</td>
<td>153</td>
<td>0.6209</td>
<td>0.3000</td>
<td>0.4104</td>
</tr>
<tr>
<td>1,2,2,1,1</td>
<td>151</td>
<td>0.6146</td>
<td>2.995</td>
<td>4.101</td>
</tr>
<tr>
<td>1,2,3,2,1</td>
<td>345</td>
<td>0.6788</td>
<td>0.3356</td>
<td>0.4486</td>
</tr>
<tr>
<td>1,3,2,2,1</td>
<td>348</td>
<td>0.6828</td>
<td>0.3346</td>
<td>0.4479</td>
</tr>
<tr>
<td>1,3,2,3,1</td>
<td>501</td>
<td>0.7090</td>
<td>0.3497</td>
<td>0.4639</td>
</tr>
<tr>
<td>1,3,3,2,1</td>
<td>489</td>
<td>0.7035</td>
<td>0.3490</td>
<td>0.4634</td>
</tr>
<tr>
<td>1,4,3,3,1</td>
<td>922</td>
<td>0.7451</td>
<td>0.3743</td>
<td>0.4914</td>
</tr>
<tr>
<td>1,3,4,3,1</td>
<td>906</td>
<td>0.7415</td>
<td>0.3750</td>
<td>0.4911</td>
</tr>
<tr>
<td>1,4,3,4,1</td>
<td>1228</td>
<td>0.7622</td>
<td>0.3878</td>
<td>0.5044</td>
</tr>
<tr>
<td>1,4,4,3,1</td>
<td>1196</td>
<td>0.7578</td>
<td>0.3855</td>
<td>0.5038</td>
</tr>
<tr>
<td>1,4,5,4,1</td>
<td>1960</td>
<td>0.7833</td>
<td>0.4065</td>
<td>0.5271</td>
</tr>
<tr>
<td>1,5,4,4,1</td>
<td>2000</td>
<td>0.7863</td>
<td>0.4067</td>
<td>0.5264</td>
</tr>
<tr>
<td>1,5,4,5,1</td>
<td>2535</td>
<td>0.7984</td>
<td>0.4166</td>
<td>0.5375</td>
</tr>
<tr>
<td>1,5,5,4,1</td>
<td>2470</td>
<td>0.7950</td>
<td>0.4158</td>
<td>0.5372</td>
</tr>
<tr>
<td>1,5,6,5,1</td>
<td>3732</td>
<td>0.8132</td>
<td>0.4332</td>
<td>0.5566</td>
</tr>
<tr>
<td>1,6,5,5,1</td>
<td>3810</td>
<td>0.8158</td>
<td>0.4337</td>
<td>0.5566</td>
</tr>
<tr>
<td>1,7,6,6,1</td>
<td>6622</td>
<td>0.8379</td>
<td>0.4574</td>
<td>0.5829</td>
</tr>
<tr>
<td>1,6,7,6,1</td>
<td>6489</td>
<td>0.8358</td>
<td>0.4561</td>
<td>0.5826</td>
</tr>
</tbody>
</table>
16 cases long lines and 25 cases of small lines (each case has a different number of machines or a different total number of cards to be allocated), only 6 times did the heuristics miss a local minimum. It should be noted that as the difference in throughput between many competing allocations is not significant, and although the state-space and the pulse heuristics pick well, there are alternate allocations that are not significantly different statistically.

It was observed that the instances in which our heuristics did not pick a local maximum were those with exponential and hyper-exponential distributions. Extensive testing indicated that it is very difficult to find cases with erlang processing times when the heuristics are off the mark! They performed better than in the case of identical exponential machines. In most real world situations, machine processing times are better modeled by erlang distributions than exponential distributions, and so our quick heuristics can be of invaluable help to a practitioner in identifying good allocations.

A sample of the results for multiple machines in a cell are shown in tables 14 - 19. Lines with symmetric cell partitions (tables 14 and 16), and asymmetric partitions (tables 15 and 17) were considered first for various different numbers of kanbans. Simultaneously, lines where the interior cell contains more machines than the end cells (tables 14 and 15) and lines where the interior cell contains fewer machines than the end cells (tables 16 and 17) were studied. Finally, lines with cells that contained wildly different number of machines were studied to see if such lack of symmetry causes our heuristics to perform badly (tables 18 and 19). In all cases, both the exponential and the erlang processing times were considered. In 19, the hyper-exponential distribution is also considered. The general strategy for a starting point is to first distribute kanbans to cells that have more than one machine in them so as to equalize their throughputs in isolation (consider each cell separately and assume no interaction among them) assuming exponential processing times, then give 1 kanban to each cell that consists of only one machine, and then distribute the remaining to the interior cells.

The heuristics were compared with with simulation over a wide variety of cases (in configurations as well as in processing times), and they did exceedingly well in picking the best allocation. In almost all cases, the allocations selected by the heuristics were a local maximum; also, further testing did not find a global maximum different from the local one. In the
Table 14: A $3/(5,8,5)/(C_1, C_2, C_3)$ line with a total of $C$ cards.

<table>
<thead>
<tr>
<th>allocation</th>
<th>states</th>
<th>pulse</th>
<th>exp</th>
<th>erl(2)</th>
<th>erl(3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C=21</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6,9,6</td>
<td>5.44</td>
<td>0.5449</td>
<td>0.5042</td>
<td>0.6058</td>
<td>0.6653</td>
</tr>
<tr>
<td>5,10,6</td>
<td>5.61</td>
<td>0.5441</td>
<td>0.5042</td>
<td>0.6013</td>
<td>0.6577</td>
</tr>
<tr>
<td>5,11,5</td>
<td>5.57</td>
<td>0.5422</td>
<td>0.4988</td>
<td>0.5966</td>
<td>0.6513</td>
</tr>
<tr>
<td>6,8,7</td>
<td>4.75</td>
<td>0.5421</td>
<td>0.4934</td>
<td>0.5904</td>
<td>0.6461</td>
</tr>
<tr>
<td>C=25</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7,11,7</td>
<td>48.13</td>
<td>0.5887</td>
<td>0.5499</td>
<td>0.6604</td>
<td>0.7226</td>
</tr>
<tr>
<td>6,13,6</td>
<td>48.40</td>
<td>0.5862</td>
<td>0.5453</td>
<td>0.6518</td>
<td>0.7108</td>
</tr>
<tr>
<td>7,10,8</td>
<td>43.65</td>
<td>0.5870</td>
<td>0.5431</td>
<td>0.6504</td>
<td>0.7119</td>
</tr>
<tr>
<td>8,9,8</td>
<td>38.15</td>
<td>0.5841</td>
<td>0.5316</td>
<td>0.6349</td>
<td>0.6934</td>
</tr>
<tr>
<td>5,14,6</td>
<td>43.93</td>
<td>0.5816</td>
<td>0.5295</td>
<td>0.6256</td>
<td>0.6772</td>
</tr>
<tr>
<td>5,15,5</td>
<td>39.03</td>
<td>0.5764</td>
<td>0.5206</td>
<td>0.6159</td>
<td>0.6679</td>
</tr>
<tr>
<td>C=30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8,14,8</td>
<td>52.39</td>
<td>0.6829</td>
<td>0.5978</td>
<td>0.7127</td>
<td>0.7729</td>
</tr>
<tr>
<td>9,12,9</td>
<td>46.55</td>
<td>0.6314</td>
<td>0.5912</td>
<td>0.7043</td>
<td>0.7654</td>
</tr>
<tr>
<td>7,16,7</td>
<td>49.91</td>
<td>0.6293</td>
<td>0.5893</td>
<td>0.6999</td>
<td>0.7583</td>
</tr>
<tr>
<td>10,10,10</td>
<td>34.59</td>
<td>0.6245</td>
<td>0.5653</td>
<td>0.6713</td>
<td>0.7288</td>
</tr>
<tr>
<td>6,18,6</td>
<td>40.26</td>
<td>0.6205</td>
<td>0.5677</td>
<td>0.6704</td>
<td>0.7255</td>
</tr>
</tbody>
</table>
Table 15: A 3/(6,10,4)/(C₁,C₂,C₃) line with a total of C cards.

<table>
<thead>
<tr>
<th>allocation</th>
<th>states</th>
<th>pulse</th>
<th>exp</th>
<th>erl(2)</th>
<th>erl(3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C=23</td>
<td>(in 10⁹)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7,11,5</td>
<td>79.96</td>
<td>0.5404</td>
<td>0.4994</td>
<td>0.5999</td>
<td>0.6587</td>
</tr>
<tr>
<td>6,12,5</td>
<td>82.13</td>
<td>0.5398</td>
<td>0.4971</td>
<td>0.5954</td>
<td>0.6517</td>
</tr>
<tr>
<td>6,13,4</td>
<td>85.41</td>
<td>0.5388</td>
<td>0.4960</td>
<td>0.5930</td>
<td>0.6497</td>
</tr>
<tr>
<td>8,10,5</td>
<td>70.92</td>
<td>0.5382</td>
<td>0.4902</td>
<td>0.5873</td>
<td>0.6432</td>
</tr>
<tr>
<td>6,11,6</td>
<td>71.37</td>
<td>0.5377</td>
<td>0.4909</td>
<td>0.5867</td>
<td>0.6429</td>
</tr>
<tr>
<td>7,10,6</td>
<td>67.00</td>
<td>0.5372</td>
<td>0.4891</td>
<td>0.5852</td>
<td>0.6411</td>
</tr>
<tr>
<td>C=30</td>
<td>(in 10¹⁰)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9,15,6</td>
<td>34.43</td>
<td>0.6069</td>
<td>0.5697</td>
<td>0.6822</td>
<td>0.7445</td>
</tr>
<tr>
<td>8,16,6</td>
<td>34.78</td>
<td>0.6063</td>
<td>0.5679</td>
<td>0.6787</td>
<td>0.7396</td>
</tr>
<tr>
<td>10,14,6</td>
<td>32.06</td>
<td>0.6059</td>
<td>0.5647</td>
<td>0.6772</td>
<td>0.7386</td>
</tr>
<tr>
<td>7,18,5</td>
<td>32.44</td>
<td>0.6018</td>
<td>0.5543</td>
<td>0.6578</td>
<td>0.7154</td>
</tr>
</tbody>
</table>

Table 16: A 3/(8,4,8)/(C₁,C₂,C₃) line with a total of C cards.

<table>
<thead>
<tr>
<th>allocation</th>
<th>states</th>
<th>pulse</th>
<th>exp</th>
<th>erl(2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C=23</td>
<td>(in 10⁹)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9,5,9</td>
<td>79.36</td>
<td>0.5406</td>
<td>0.5007</td>
<td>0.6030</td>
</tr>
<tr>
<td>9,6,8</td>
<td>73.82</td>
<td>0.5382</td>
<td>0.4957</td>
<td>0.5951</td>
</tr>
<tr>
<td>10,4,9</td>
<td>75.97</td>
<td>0.5395</td>
<td>0.4926</td>
<td>0.5922</td>
</tr>
<tr>
<td>8,7,8</td>
<td>65.10</td>
<td>0.5343</td>
<td>0.4911</td>
<td>0.5883</td>
</tr>
<tr>
<td>C=30</td>
<td>(in 10¹¹)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12,6,12</td>
<td>32.52</td>
<td>0.6067</td>
<td>0.5678</td>
<td>0.6820</td>
</tr>
<tr>
<td>11,8,11</td>
<td>30.00</td>
<td>0.6041</td>
<td>0.5660</td>
<td>0.6768</td>
</tr>
<tr>
<td>13,5,12</td>
<td>30.55</td>
<td>0.6051</td>
<td>0.5585</td>
<td>0.6707</td>
</tr>
<tr>
<td>13,4,13</td>
<td>26.80</td>
<td>0.6013</td>
<td>0.5381</td>
<td>0.6364</td>
</tr>
<tr>
<td>10,10,10</td>
<td>22.51</td>
<td>0.5960</td>
<td>0.5533</td>
<td>0.6349</td>
</tr>
<tr>
<td>9,12,9</td>
<td>14.21</td>
<td>0.5834</td>
<td>0.5339</td>
<td>0.6249</td>
</tr>
<tr>
<td>8,14,8</td>
<td>7.63</td>
<td>0.5667</td>
<td>0.5094</td>
<td>0.6040</td>
</tr>
</tbody>
</table>
Table 17: A $3/(8,4,6)/(C_1, C_2, C_3)$ line with a total of $C$ cards.

<table>
<thead>
<tr>
<th>allocation</th>
<th>states</th>
<th>pulse</th>
<th>exp</th>
<th>erl(2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C=21$ (in $10^9$)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9,5,7</td>
<td>5.56</td>
<td>0.5450</td>
<td>0.5056</td>
<td>0.6079</td>
</tr>
<tr>
<td>9,6,6</td>
<td>5.30</td>
<td>0.5426</td>
<td>0.5008</td>
<td>0.5997</td>
</tr>
<tr>
<td>10,5,6</td>
<td>5.44</td>
<td>0.5437</td>
<td>0.4981</td>
<td>0.5968</td>
</tr>
<tr>
<td>10,4,7</td>
<td>5.33</td>
<td>0.5437</td>
<td>0.4976</td>
<td>0.5966</td>
</tr>
<tr>
<td>8,6,7</td>
<td>5.17</td>
<td>0.5424</td>
<td>0.4988</td>
<td>0.5964</td>
</tr>
<tr>
<td>9,4,8</td>
<td>5.16</td>
<td>0.5431</td>
<td>0.4959</td>
<td>0.5943</td>
</tr>
<tr>
<td>8,7,6</td>
<td>4.67</td>
<td>0.5383</td>
<td>0.4946</td>
<td>0.5915</td>
</tr>
<tr>
<td>$C=30$ (in $10^{10}$)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13,7,10</td>
<td>51.29</td>
<td>0.6329</td>
<td>0.5980</td>
<td>0.7126</td>
</tr>
<tr>
<td>13,8,9</td>
<td>50.22</td>
<td>0.6318</td>
<td>0.5966</td>
<td>0.7108</td>
</tr>
<tr>
<td>14,7,9</td>
<td>50.74</td>
<td>0.6322</td>
<td>0.5942</td>
<td>0.7087</td>
</tr>
<tr>
<td>12,8,10</td>
<td>49.43</td>
<td>0.6316</td>
<td>0.5962</td>
<td>0.7081</td>
</tr>
<tr>
<td>14,6,10</td>
<td>49.74</td>
<td>0.6321</td>
<td>0.5925</td>
<td>0.7072</td>
</tr>
<tr>
<td>13,6,11</td>
<td>48.71</td>
<td>0.6317</td>
<td>0.5925</td>
<td>0.7055</td>
</tr>
<tr>
<td>12,6,12</td>
<td>45.18</td>
<td>0.6298</td>
<td>0.5866</td>
<td>0.6963</td>
</tr>
<tr>
<td>14,8,8</td>
<td>47.86</td>
<td>0.6300</td>
<td>0.5853</td>
<td>0.6964</td>
</tr>
<tr>
<td>15,7,8</td>
<td>47.21</td>
<td>0.6298</td>
<td>0.5813</td>
<td>0.6923</td>
</tr>
</tbody>
</table>
Table 18: A 4/(4,3,5,2)/(C_1, C_2, C_3, C_4) line with a total of 17 cards.

<table>
<thead>
<tr>
<th>allocation</th>
<th>states</th>
<th>pulse</th>
<th>exp</th>
<th>erl(2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4,4,7,2</td>
<td>19,646,550</td>
<td>0.5518</td>
<td>0.5083</td>
<td>0.6078</td>
</tr>
<tr>
<td>4,5,6,2</td>
<td>18,614,610</td>
<td>0.5501</td>
<td>0.5073</td>
<td>0.6063</td>
</tr>
<tr>
<td>4,3,8,2</td>
<td>18,191,250</td>
<td>0.5478</td>
<td>0.4936</td>
<td>0.5894</td>
</tr>
<tr>
<td>5,4,6,2</td>
<td>19,289,340</td>
<td>0.5535</td>
<td>0.5089</td>
<td>0.6113</td>
</tr>
<tr>
<td>5,3,7,2</td>
<td>18,877,320</td>
<td>0.5516</td>
<td>0.5044</td>
<td>0.6060</td>
</tr>
<tr>
<td>5,4,7,1</td>
<td>19,145,280</td>
<td>0.5435</td>
<td>0.4625</td>
<td>0.4915</td>
</tr>
<tr>
<td>3,6,6,2</td>
<td>15,505,560</td>
<td>0.5396</td>
<td>0.4785</td>
<td>0.5551</td>
</tr>
<tr>
<td>3,5,7,2</td>
<td>17,276,490</td>
<td>0.5437</td>
<td>0.4782</td>
<td>0.5543</td>
</tr>
<tr>
<td>4,4,6,3</td>
<td>17,301,900</td>
<td>0.5515</td>
<td>0.5058</td>
<td>0.6054</td>
</tr>
<tr>
<td>4,3,7,3</td>
<td>16,701,300</td>
<td>0.5493</td>
<td>0.4932</td>
<td>0.5887</td>
</tr>
<tr>
<td>6,4,5,2</td>
<td>16,537,500</td>
<td>0.5491</td>
<td>0.4901</td>
<td>0.5858</td>
</tr>
</tbody>
</table>

Table 19: A 6/(4,3,2,5,1,5)/(C_1, C_2, C_3, C_4, C_5, C_6) line with a total of 31 cards.

<table>
<thead>
<tr>
<th>allocation</th>
<th>states, in 10^{11}</th>
<th>pulse</th>
<th>hyperl</th>
<th>exp</th>
<th>erl(2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6,5,4,8,1,7</td>
<td>11.69</td>
<td>0.3237</td>
<td>0.6070</td>
<td>0.5565</td>
<td>0.6716</td>
</tr>
<tr>
<td>6,5,4,7,2,7</td>
<td>10.93</td>
<td>0.3128</td>
<td>0.6068</td>
<td>0.5560</td>
<td>0.6711</td>
</tr>
<tr>
<td>6,6,3,8,1,7</td>
<td>11.63</td>
<td>0.3225</td>
<td>0.6065</td>
<td>0.5530</td>
<td>0.6686</td>
</tr>
<tr>
<td>7,4,3,9,1,7</td>
<td>11.20</td>
<td>0.3212</td>
<td>0.6058</td>
<td>0.5507</td>
<td>0.6655</td>
</tr>
<tr>
<td>7,4,3,8,1,8</td>
<td>11.14</td>
<td>0.3189</td>
<td>0.6066</td>
<td>0.5484</td>
<td>0.6640</td>
</tr>
<tr>
<td>8,4,3,8,1,7</td>
<td>10.21</td>
<td>0.3177</td>
<td>0.6044</td>
<td>0.5451</td>
<td>0.6589</td>
</tr>
</tbody>
</table>
cases (very rare) where the heuristics did not select the best allocation, they were among the top three and did not differ significantly from the best. On the whole, simply checking the allocations with highest pulse and state-space is very satisfactory. Taking the top three always included the best allocation we found.

Other experiments were conducted with two and three cells and varying number of machines and kanbans. The results obtained were similar. Almost always, the heuristics picked the best allocation. In the range where both of the heuristics were expected to do well, they coincided in picking the right allocation. With erlang(2) and erlang(3) machines, the heuristics were even more accurate. In those cases when they picked right for the exponential processing times, they were also right for the erlangs, and when they missed (always very narrowly) for the exponentials, they were correct for the erlangs!

A summary of the computational results (one machine/cell included) is given in the table 20. Far more simulations were done, but in many cases only a subset of the processing time distributions were considered, or the local minimum check was done only for a subset of the processing time distributions. Various cell partitions were considered with machines that had exponential processing times to verify how well the heuristics performed. These include 18/(6,6,6) (/), 18/(9,9)/ () , 20 /(5,5,5,5) /() , 20 /(10,10) /(), 20 /(4,6,6,4) /(), 24 /(6,6,6,6) /() and other such partitions. The results were are similar—the heuristics do well at appropriate ranges (state-space at small ratios of total kanbans to total cells, and pulse at higher ratios). In the table below, only those simulations are considered in which all the processing time distributions were evaluated. In each box, we give the total number of cases simulated in this category, the number of cases when the heuristic picked the best, the number of cases when one of the top three was picked and the maximum % error.
Table 20: Summary of computational work. Each box contains total #, # best picked, # in top three, max. error.

<table>
<thead>
<tr>
<th>proc.time distribution</th>
<th>4-7 m/c</th>
<th>12-18 m/c</th>
<th>20-30 m/c</th>
</tr>
</thead>
<tbody>
<tr>
<td>hyper exponential</td>
<td>10,9,9,0.5</td>
<td>10,9,9,0.5</td>
<td>10,8,9,0.5</td>
</tr>
<tr>
<td>exponential</td>
<td>10,10,10,0.0</td>
<td>10,9,9,0.4</td>
<td>10,8,9,0.4</td>
</tr>
<tr>
<td>erlang(2)</td>
<td>10,10,10,0.0</td>
<td>10,10,10,0.0</td>
<td>10,9,10,0.4</td>
</tr>
<tr>
<td>erlang(3)</td>
<td>10,10,10,0.0</td>
<td>10,9,10,0.4</td>
<td>10,10,10,0.0</td>
</tr>
<tr>
<td>erlang(k)</td>
<td>10,10,10,0.0</td>
<td>10,10,10,0.0</td>
<td>10,10,10,0.0</td>
</tr>
</tbody>
</table>

References


