A Markov Chain Monte Carlo Approach to Cost Matrix Generation for Scheduling Performance Evaluation

Louis-Claude Canon[1,2], Mohamad El Sayah[2], and Pierre-Cyrille Héam[2]

[1]École Normale Supérieure de Lyon, CNRS & Inria, France
[2] FEMTO-ST Institute, CNRS, Univ. Bourgogne Franche-Comté, France,

November 9, 2018

Abstract

In high performance computing, scheduling of tasks and allocation to machines is very critical especially when we are dealing with heterogeneous execution costs. Simulations can be performed with a large variety of environments and application models. However, this technique is sensitive to bias when it relies on random instances with an uncontrolled distribution. We use methods from the literature to provide formal guarantee on the distribution of the instance. In particular, it is desirable to ensure a uniform distribution among the instances with a given task and machine heterogeneity. In this article, we propose a method that generates instances (cost matrices) with a known distribution where tasks are scheduled on machines with heterogeneous execution costs.

1 Introduction

Empirical assessment is critical to determine the best scheduling heuristics on any parallel platform. However, the performance of any heuristic may be specific to a given parallel computer. In addition to experimentation on real platforms, simulation is an effective tool to quantify the quality of scheduling heuristics. Even though simulations provide weaker evidence, they can be performed with a large variety of environments and application models, resulting in broader conclusions. However, this technique is sensitive to bias when it relies on random instances with an uncontrolled or irrelevant distribution. For instance, in uniformly distributed random graphs, the probability that the diameter is 2 tends exponentially to 1 as the size of the graph tends to infinity [1]. Even though such instances may be sometimes of interest, they prove useless in most practical contexts. We propose a method that generates instances with a known distribution for a set of classical problems where tasks must be scheduled on machines (or processors) with heterogeneous execution costs. This is critical to the empirical validation of many new heuristics like BalSuff [2] for the problem $R||C_{max}$ and PEFT [3] for $R|\text{prec}|C_{\max}$ in Graham's notation [4].

In this context, an instance consists of a $n \times m$ cost matrices, M, where the element of row i and column j, $M(i,j)$, represents the execution cost of task i on machine j. Like the diameter for graphs, multiple criteria characterize cost matrices. First, the heterogeneity can be determined globally with the variance of all costs, but also relatively to the rows or columns. For instance, the dispersion of the means on each row, which corresponds to the varying costs for each task, impacts
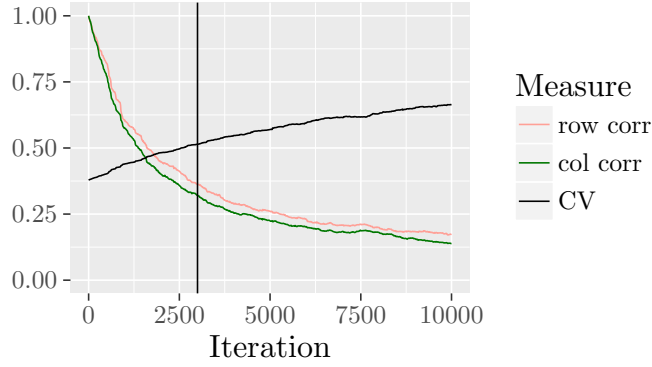
Figure 1: Cost Coefficient-of-Variation (ratio of standard deviation to mean) and mean row and column correlations at each iteration of the shuffling method [5] when generating a 100×30 cost matrix. The shuffling method arbitrarily stops after $3\,000$ iterations (represented by the black vertical line).

the performance of some scheduling heuristics [5]. The correlations between the rows and columns also play an important role as it corresponds to the machines being either related or specialized, with some affinity between the tasks and the machines [6].

Among existing methods, the shuffling one [5] starts by an initial matrix in which rows are proportional to each other (leading to large row and column correlations). Then, it proceeds to mix the values in the matrix such as to keep the same sum on each row and column. This ensures that the row and column heterogeneity remains stable, while the correlation decreases. However, this approach is heuristic and provides no formal guarantee on the distribution of the instances. In addition, when the number of shuffles increases, the cost CV increases, which leads to non-interpretable results (see Figure 1).

While other methods exist, some of them with stronger formal guarantees, it remains an open problem to ensure a uniform distribution among the instances that have a given task and machine heterogeneity. Our contribution is to control the row and column heterogeneity, while limiting the overall variance and ensuring a uniform distribution among the set of possible instances. The approach is based on a Markov Chain Monte Carlo process and relies on contingency tables[1]. More precisely, the proposed random generation process is based on two steps. For a given n (number of tasks), m (number of machines) and N (sum of the cost of the tasks):

1. Randomly generate the average cost of each task and the average speed of each machine. This random generation is performed uniformly using classical recursive algorithms [7]. In order to control the heterogeneity, we show how to restrict this uniform random generation to interesting classes of vectors. This step is described in Section 3.

2. Next, the cost matrices can be generated using a classical MCMC approach: from an initial matrix, a random walk in the graph of contingency tables is performed. It is known (see for instance [8]) that if the Markov Chain associated with this walk is ergodic and symmetric, then the unique stationary distribution exists and is uniform. Walking enough steps in the

---

[1]A contingency table is a matrix with the sum of each row (resp. column) displayed in an additional total row (resp. column). They are usually used to show the distribution of two variables.

graph leads to any state with the same probability. Section 4 provides several symmetric and ergodic Markov Chains for this problem. The main contribution of this section is to extend known results for contingency tables to contingency tables with min/max constraints.

In order to evaluate the mixing time of the proposed Markov Chains (the mixing time is the number of steps to walk in order to be close to the uniform distribution), we propose practical and statistical estimations in Section 5. Note that obtaining theoretical bound on mixing time is a very hard theoretical problem, still open in the general case of unconstrained contingency tables. In Section 6, we used our random generation process to evaluate scheduling algorithms. The algorithms are implemented in R and Python and the related code, data and analysis are available in [9].

2 Related Work

Two main methods have been used in the literature: RB (range-based) and CVB (Coefficient-of-Variation-Based) [10, 11]. Both methods follow the same principle: n vectors of m values are first generated using a uniform distribution for RB and a gamma distribution for CVB; then, each row is multiplied by a random value using the same distribution for each method. A third optional step consists in sorting each row in a submatrix, which increases the correlation of the cost matrix. However, these methods are difficult to use when generating a matrix with given heterogeneity and low correlation [5, 6].

More recently, two additional methods have been proposed for a better control of the heterogeneity: SB (shuffling-based) and NB (noise-based) [5]. In the first step of SB, one column of size n and one row of size m are generated using a gamma distribution. These two vectors are then multiplied to obtain a $n \times m$ cost matrix with a strong correlation. To reduce it, values are shuffled without changing the sum on any row or column as it is done is Section 4: selecting four elements on two distinct rows and columns (a submatrix of size 2×2); and, removing/adding the maximum quantity to two elements on the same diagonal while adding/removing the same quantity to the last two elements on the other diagonal. While NB shares the same first step, it introduces randomness in the matrix by multiplying each element by a random variable with expected value one instead of shuffling the elements. When the size of the matrix is large, SB and NB provide some control on the heterogeneity but the distribution of the generated instances is unknown.

Finally, CNB (correlation noise-based) and CB (combination-based) have been proposed to control the correlation [6]. CNB is a direct variation of CB to specify the correlation more easily. CB combines correlated matrices with an uncorrelated one to obtain the desired correlation. As for SB and NB, both methods have asymptotic guarantees when the size of the matrix tends to infinity, but no guarantee on how instances are distributed.

The present work relies on contingency tables/matrices, which are important data structures used in statistics for displaying the multivariate frequency distribution of variables, introduced in 1904 by K. Pearson [12]. The MCMC approach is the most common way used in the literature for the uniform random generation of contingency tables (see for instance [13, 14]). Mixing time results have been provided for the particular case of $2 \times n$ sized tables in [15] and the latter using a coupling argument in [16]. In this restricted context a divide-and-conquer algorithm has recently been pointed out [17]. In practice, there are MCMC dedicated packages for most common programming languages: `mcmc`[2] for R, `pymc`[3] for Python, . . .

---

[2] https://cran.r-project.org/web/packages/mcmc/index.html
[3] https://pypi.python.org/pypi/pymc/

More generally, random generation is a natural way for performance evaluation used, for instance in SAT-solver competitions[4]. In a distributed computing context, it has been used for instance for the random generation of DAG modelling graph task for parallel environments [18, 19].

3 Contingency vectors initialization

Considering n tasks and m machines, the first step in order to generate instances is to fix the average cost of each task and the average speed of each machine. Since n and m are fixed, instead of generating cost averages, we generate the sum of the cost on each row and column, which is related. The problem becomes, given n, m and N (total cost) to generate randomly (and uniformly) two vectors $\overline{\mu} \in \mathbb{N}^n$ and $\overline{\nu} \in \mathbb{N}^m$ satisfying:

$$\sum_{i=1}^{n} \overline{\mu}(i) = \sum_{j=1}^{m} \overline{\nu}(j) = N, \tag{1}$$

with the following convention on notations: for any vector $\overline{v} = (v_1, \ldots, v_\ell) \in \mathbb{N}^\ell$, v_i is denoted $\overline{v}(i)$.

Moreover, the objective is also to limit the maximum value. This is useful to avoid large variance: for this purpose we restrict the generation to vectors whose parameters are in a controlled interval $[\alpha, \beta]$. This question is addressed in this section using a classical recursive approach [7]. More precisely, let $\alpha \leq \beta$ be positive integers and $H_{N,n}^{\alpha,\beta}$ be the subset of elements $\overline{\mu}$ of $\mathbb{N}^n$ such that $N = \sum_{i=1}^{n} \overline{\mu}(i)$ and for all $1 \leq i \leq n$, $\alpha \leq \overline{\mu}(i) \leq \beta$ (i.e. the set of all possible vectors with values between α and β). Let $h_{N,n}^{\alpha,\beta}$ be the cardinal of $H_{N,n}^{\alpha,\beta}$. By decomposition one has

$$h_{N,n}^{\alpha,\beta} = \sum_{k=\alpha}^{\beta} h_{N-k,n-1}^{\alpha,\beta}. \tag{2}$$

Moreover,

$$\begin{aligned} &h_{N,n}^{\alpha,\beta} = 0 \text{ if } \alpha n < N \text{ or } \beta n > N \text{ and,} \\ &h_{N,1}^{\alpha,\beta} = 1 \text{ if } \alpha < N < \beta. \end{aligned} \tag{3}$$

Algorithm 1 uniformly generates a random vector over $H_{N,n}^{\alpha,\beta}$.

Note that integers involved in these computations may become rapidly very large. Working with floating point approximations to represent integers may be more efficient. Moreover, with the rounded errors the random generation stays very close to the uniform distribution [20].

Figure 2 depicts the distribution of the values when varying the interval $[\alpha, \beta]$ for $n = 10$ and $N = 100$. Without constraint ($\alpha = 0$ and $\beta = 100$), the distribution is similar to an exponential one: small values are more likely to appear in a vector than large ones. When only the largest value is bounded ($\alpha = 0$ and $\beta = 15$), then the shape of the distribution is inverted with smaller values being less frequent. Finally, bounding from both sides ($\alpha = 5$ and $\beta = 15$) leads to a more uniform distribution.

Figure 3 shows the CV obtained for all possible intervals $[\alpha, \beta]$. The more constrained the values are, the lower the CV. The CV goes from 0 when either $\alpha = 10$ or $\beta = 10$ (the vector contains only the value 10) to 1 when $\alpha = 0$ and $\beta = 100$.

---

[4]http://www.satcompetition.org/

Algorithm 1: Generate Sequences

Input: Integers N, n, α, β
Output: $\overline{\mu} \in \mathbb{N}^n$ such that $\alpha \leq \overline{\mu}(i) \leq \beta$ and $\sum_i \overline{\mu}(i) = N$ if it is possible
$\quad\quad\quad\quad \perp$ otherwise.

1 begin
2 $\quad$ **if** $\alpha > \beta$ **or** $n\alpha > N$ **or** $n\beta < N$ **then**
3 $\quad\quad$ **return** $\perp$
4 $\quad$ **for** $1 \leq k \leq n$ **and** $0 \leq N' \leq N$ **do**
5 $\quad\quad$ compute $h_{N',k}^{\alpha,\beta}$ using **(2)** and **(3)**.
6 $\quad$ **for** $i \in [1, \ldots, n]$ **do**
7 $\quad\quad$ $s = 0$
8 $\quad\quad$ **if** $N - s \geq 0$ **then**
9 $\quad\quad\quad$ **pick at random** $\overline{\mu}(i) \in [\alpha, \beta]$ **with** $\mathbb{P}(\overline{\mu}(i) = k) = \dfrac{h_{N-s-k,k}^{\alpha,\beta}}{h_{N-s,n-i}^{\alpha,\beta}}$
10 $\quad\quad\quad$ $s = s + k$
11 $\quad\quad$ **else**
12 $\quad\quad\quad$ $\overline{\mu}_i = 0$
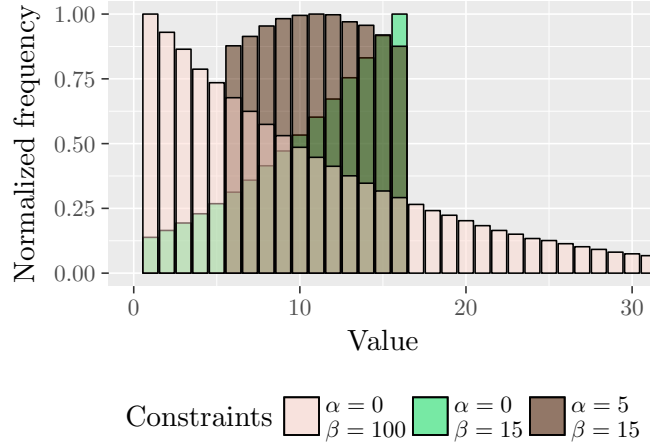**13** $\quad$ **return** $\overline{\mu}$

Figure 2: Frequency of each value in a vector of size $n = 10$ with $N = 100$ generated by Algorithm 1 for three combinations of constraints for the minimum α and maximum β. For each case, the frequencies were determined by generating 100 000 vectors and are normalized to the maximum frequency. The frequency for large values when $\alpha = 0$ and $\beta = 100$ are not shown.
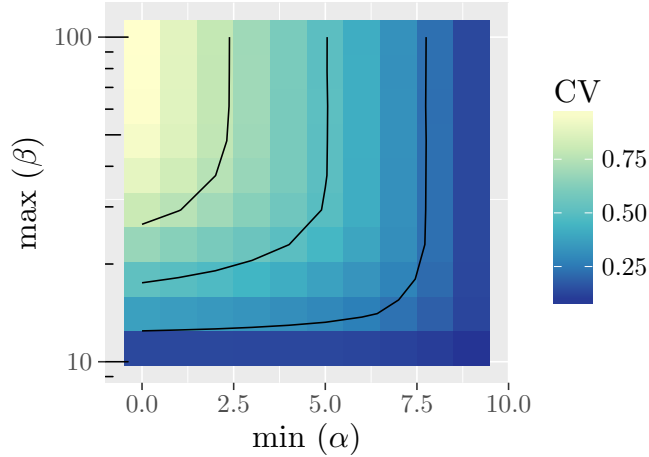
Figure 3: Mean CV in vectors of size $n = 10$ with $N = 100$ generated by Algorithm 1 for different constraints for the minimum α and maximum β. Each tile corresponds to $10\,000$ vectors. The contour lines correspond to the levels in the legend (2.5, 5, 7.5 and 10).

It is also possible to generate uniform vectors using Boltzmann samplers [21]: this approach consists in generating each $\bar{\nu}(i)$ independently according to an exponential law of parameter γ. Theoretical results of [21] show that by choosing the right γ, the sum of the generated $\bar{\nu}(i)$'s is close to N with a high probability. In order to get precisely N, it suffices to use a rejection approach. This is consistent with the seemingly exponential distribution in Figure 2 in the unconstrained case. Moreover, in this case, Figure 3 shows that the CV is close to one, which is also the CV of an exponential distribution.

4   Symmetric Ergodic Markov Chains for the Random Generation

We can now generate two random vectors $\bar{\mu}$ and $\bar{\nu}$ containing the sum of each row and column with Algorithm 1. To obtain actual cost, we use Markov Chains to generate the corresponding contingency table. Random generation using finite discrete Markov Chains can easily be explained using random walk on finite graphs. Let Ω be the finite set of all possible cost matrices (also called states) with given row and column sums: we want to sample uniformly one of its elements. However, Ω is too large to be built explicitly. The approach consists in building a directed graph whose set of vertices is Ω and whose set of edges represent all the possible transitions between any pair of states. Each edge of the graph is weighted by a probability with a classical normalization: for each vertex, the sum of the probabilities on outgoing edges is equal to 1. One can now consider random walks on this graph. A classical Markov Chain result claims that for some families of probabilistic graphs/Markov Chains, walking long enough in the graph, we have the same probability to be in each state, whatever the starting vertex of the walk [8, Theorem 4.9].

This is the case for symmetric ergodic Markov Chains [8, page 37]. Symmetric means that if there is an edge (x, y) with probability p, then the graph has an edge (y, x) with the same probability. A Markov Chain is ergodic if it is aperiodic (the gdc of the lengths of loops of the graph is 1) and if the graph is strongly connected. When there is a loop of length 1, the ergodicity issue reduces
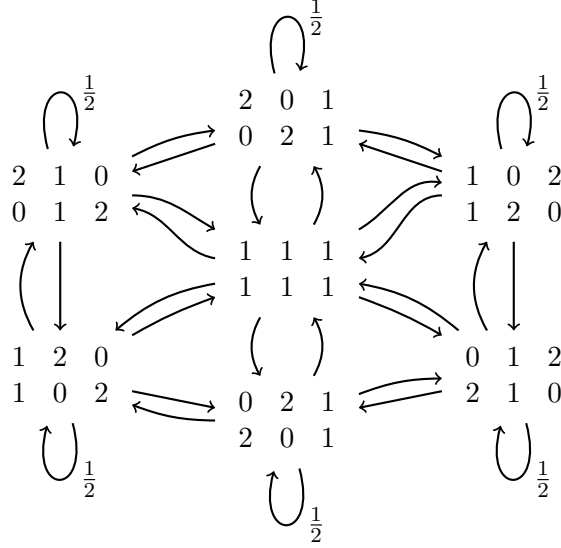
6

Figure 4: Example of the underlying graph of a Markov Chain when the sum of each row is three and the sum of column is two. Unless otherwise stated, each transition probability is $\frac{1}{6}$.

to the strongly connected problem. In general, the graph is not explicitly built and neighborhood relation is defined by a function, called a random mapping, on each state. For a general reference on finite Markov Chains with many pointers, see [8].

An illustration example is depicted on Fig 4. For instance, starting arbitrarily from the central vertex, after one step, we are in any other vertex with probability $\frac{1}{6}$ (and with probability 0 in the central vertex since there is no self-loop on it). After two steps, we are in the central vertex with probability $\frac{1}{6}$ and in any other with probability $\frac{5}{36}$. In this simple example, one can show that after $n+1$ step, the probability to be in the central node is $p_{n+1} = \frac{1}{7}(1 - \left(\frac{-1}{6}\right)^n)$ and is $\frac{1-p_{n+1}}{6}$ for all the other nodes. All probabilities tends to $\frac{1}{7}$ when n grows.

This section is dedicated to building symmetric and ergodic Markov Chains for our problem. In Section 4.1 we define the sets Ω that are interesting for cost matrices. In Section 4.2, Markov Chains are proposed using a dedicated random mapping and are proved to be symmetric and ergodic. Finally, in Section 4.3 we use classical techniques to transform the Markov Chains into other symmetric ergodic MC mixing faster (i.e. the number of steps required to be close to the uniform distribution is smaller).

Recall that N, n, m are positive integers and that $\overline{\mu} \in \mathbb{N}^n$ and $\overline{\nu} \in \mathbb{N}^m$ satisfy Equation (1).

4.1 Contingency Tables

In this section, we define the state space of the Markov Chains. We consider contingency tables with fixed sums on rows and columns. We also introduce min/max constraints in order to control the variance of the value. We denote by $\Omega_{n,m}^N(\overline{\mu}, \overline{\nu})$ the set of positive $n \times m$ matrices M over $\mathbb{N}$ such that for every $i \in \{1, \ldots, n\}$ and every $j \in \{1, \ldots, m\}$,

$$\sum_{k=1}^{m} M(i,k) = \overline{\mu}(i) \quad \text{and} \quad \sum_{k=1}^{n} M(k,j) = \overline{\nu}(j) \tag{4}$$

7

For example, the matrix

$$M_{\mathrm{exa}} = \begin{pmatrix} 3 & 1 \\ 2 & 0 \\ 5 & 10 \end{pmatrix}$$

is in $\Omega_{2,3}(\overline{\mu}_{\mathrm{exa}}, \overline{\nu}_{\mathrm{exa}})$, where $\overline{\mu}_{\mathrm{exa}} = (4, 2, 15)$ and $\overline{\nu}_{\mathrm{exa}} = (10, 11)$.

The first restriction consists in having a global minimal value α and a maximal global value β on the considered matrices. Let α, β be positive integers. We denote by $\Omega_{n,m}^{N}(\overline{\mu}, \overline{\nu})[\alpha, \beta]$ the subset of $\Omega_{n,m}^{N}(\overline{\mu}, \overline{\nu})$ of matrices M such that for all i, j, $\alpha \le M(i,j) \le \beta$. For example, $M_{\mathrm{exa}} \in \Omega_{2,3}(\overline{\mu}_{\mathrm{exa}}, \overline{\nu}_{\mathrm{exa}})[0, 12]$. If $\beta < \alpha$, then $\Omega_{n,m}^{N}(\overline{\mu}, \overline{\nu})[\alpha, \beta] = \emptyset$. Moreover, according to Equation (4), one has

$$\begin{aligned} \Omega_{n,m}^{N}(\overline{\mu}, \overline{\nu}) &= \Omega_{n,m}^{N}(\overline{\mu}, \overline{\nu})[0, N] \\ &= \Omega_{n,m}^{N}(\overline{\mu}, \overline{\nu})[0, \min(\max_{1 \le k \le m} \overline{\mu}(k), \\ &\qquad \max_{1 \le k \le n} \overline{\nu}(k))]. \end{aligned} \tag{5}$$

Now we consider min/max constraints on each row and each line. Let $\overline{\alpha}_c, \overline{\beta}_c \in \mathbb{N}^m$ and $\overline{\alpha}_r, \overline{\beta}_r \in \mathbb{N}^n$. We denote by $\Omega_{n,m}^{N}(\overline{\mu}, \overline{\nu})[\overline{\alpha}_c, \overline{\beta}_c, \overline{\alpha}_r, \overline{\beta}_r]$ the subset of $\Omega_{n,m}^{N}(\overline{\mu}, \overline{\nu})$ of matrices M satisfying: for all i, j, $\overline{\alpha}_c(j) \le M(i,j) \le \overline{\beta}_c(j)$ and $\overline{\alpha}_r(i) \le M(i,j) \le \overline{\beta}_r(i)$. For instance,

$$M_{\mathrm{exa}} \in \Omega_{2,3}(\overline{\mu}_{\mathrm{exa}}, \overline{\nu}_{\mathrm{exa}})[(1, 0, 5), (3, 2, 10), (2, 0), (5, 10)].$$

Using Equation (4), one has for every $\alpha, \beta \in \mathbb{N}$,

$$\begin{aligned} \Omega_{n,m}^{N}(\overline{\mu}, \overline{\nu})[\alpha, \beta] &= \Omega_{n,m}^{N}(\overline{\mu}, \overline{\nu})[(\alpha, \dots, \alpha), \\ &\quad (\beta, \dots, \beta), (\alpha, \dots, \alpha), (\beta, \dots, \beta)]. \end{aligned} \tag{6}$$

To finish, the more general constrained case, where min/max are defined for each element of the matrices. Let $A_{\min}$ and $B_{\max}$ be two $n \times m$ matrices of positive integers. We denote by $\Omega_{n,m}^{N}(\overline{\mu}, \overline{\nu})[Am, Bm]$ the subset of $\Omega_{n,m}^{N}(\overline{\mu}, \overline{\nu})$ of matrices M such that for all i, j, $A_{\min}(i,j) \le M(i,j) \le B_{\max}(i,j)$. For instance, one has $M_{\mathrm{exa}} \in \Omega_{n,m}^{N}(\overline{\mu}, \overline{\nu})[A_{\mathrm{exa}}, B_{\mathrm{exa}}]$, with

$$A_{\mathrm{exa}} = \begin{pmatrix} 3 & 2 & 4 \\ 0 & 0 & 5 \end{pmatrix} \quad \text{and} \quad B_{\mathrm{exa}} = \begin{pmatrix} 5 & 4 & 6 \\ 1 & 3 & 12 \end{pmatrix}.$$

For every $\overline{\alpha}_c, \overline{\beta}_c \in \mathbb{N}^m, \overline{\alpha}_r, \overline{\beta}_r \in \mathbb{N}^n$, one has

$$\Omega_{n,m}^{N}(\overline{\mu}, \overline{\nu})[\overline{\alpha}_c, \overline{\beta}_c, \overline{\alpha}_r, \overline{\beta}_r] = \Omega_{n,m}^{N}(\overline{\mu}, \overline{\nu})[A, B], \tag{7}$$

where $A(i,j) = \max\{\overline{\alpha}_c(j), \overline{\alpha}_r(i)\}$ and $B(i,j) = \min\{\overline{\beta}_c(j), \overline{\beta}_r(i)\}$.

4.2 Markov Chains

As explained before, the random generation process is based on symmetric ergodic Markov Chains. This section is dedicated to define such chains on state spaces of the form $\Omega_{n,m}^{N}(\overline{\mu}, \overline{\nu})$, $\Omega_{n,m}^{N}(\overline{\mu}, \overline{\nu})[\alpha, \beta]$, $\Omega_{n,m}^{N}(\overline{\mu}, \overline{\nu})[\overline{\alpha}_c, \overline{\beta}_c, \overline{\alpha}_r, \overline{\beta}_r]$ and $\Omega_{n,m}^{N}(\overline{\mu}, \overline{\nu})[A_{\min}, B_{\max}]$. According to Equations (5), (6) and (7), it suffices to work on $\Omega_{n,m}^{N}(\overline{\mu}, \overline{\nu})[A_{\min}, B_{\max}]$. To simplify the notation, let us denote by Ω the set $\Omega_{n,m}^{N}(\overline{\mu}, \overline{\nu})[A_{\min}, B_{\max}]$.

For any $1 \leq i_0, i_1 \leq n$, any $1 \leq j_0, j_1, \leq m$, such that $i_0 \neq i_1$ and $j_0 \neq j_1$, we denote by Δ_{i_0,i_1,j_0,j_1} the $n \times m$ matrix defined by $\Delta(i_0, j_0) = \Delta(i_1, j_1) = 1$, $\Delta(i_0, j_1) = \Delta(i_1, j_0) = -1$, and $\Delta(i, j) = 0$ otherwise. For instance, for $n = 3$ and $m = 4$ one has

$$\Delta_{1,2,1,3} = \begin{pmatrix} 1 & 0 & \text{-}1 & 0 \\ \text{-}1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

Tuple (i_0, j_0, i_1, j_1) is used as follow to shuffle a cost matrix and to transit from one state to another in the markov chain: Δ_{i_0,i_1,j_0,j_1} is added to the current matrix, which preserves the row and column sums. Formally, let $K = \{(i_0, j_0, i_1, j_1) \mid i_0 \neq i_1, j_0 \neq j_1, 1 \leq i_0, i_1 \leq n, 1 \leq j_0, j_1 \leq m\}$ be the set of all possible tuples. Let f be the mapping function from $\Omega \times K$ to Ω defined by $f(M, (i_0, j_0, i_1, j_1)) = M + \Delta_{(i_0,j_0,i_1,j_1)}$ if $M + \Delta_{(i_0,j_0,i_1,j_1)} \in \Omega$ and M otherwise. The mapping is called at each iteration, changing the instance until it is sufficiently shuffled.

We consider the Markov chain $\mathcal{M}$ defined on Ω by the random mapping $f(\cdot, U_K)$, where U_K is a uniform random variable on K.

The following result gives the properties of the markov chain and is an extension of a similar result [13] on $\Omega_{n,m}^N(\overline{\mu}, \overline{\nu})$. The difficulty is to prove that the underlying graph is strongly connected since the constraints are hindering the moves.

Theorem 1. *The Markov Chain $\mathcal{M}$ is symmetric and ergodic.*

The proof of Theorem 1 is based on Lemma 3 and 4.

Definition 2. *Let A et B be two elements of Ω. A finite sequence $u_1 = (i_1, j_1), \ldots, u_r = (i_r, j_r)$ of pairs of indices in $\{1, \ldots, n\} \times \{1, \ldots, m\}$ is called a stair sequence for A and B if it satisfies the following properties:*

1. *$r \geq 4$,*

2. *If $k \neq \ell$, then $u_k \neq u_\ell$,*

3. *If $1 \leq k < r$ is even, then $j_k = j_{k+1}$ and $A(i_k, j_k) < B(i_k, j_k)$*

4. *If $1 \leq k < r$ is odd, then $i_k = i_{k+1}$ and $A(i_k, j_k) > B(i_k, j_k)$,*

5. *r is even and $j_r = j_1$,*

Consider, for instance, the matrices

$$A_1 = \begin{pmatrix} 3 & 0 & 0 & 0 & 7 \\ 7 & 4 & 0 & 0 & 0 \\ 0 & 7 & 5 & 0 & 0 \\ 0 & 0 & 7 & 6 & 0 \\ 0 & 0 & 0 & 7 & 5 \end{pmatrix} \quad B_1 = \begin{pmatrix} 2 & 1 & 0 & 0 & 7 \\ 7 & 3 & 1 & 0 & 0 \\ 0 & 7 & 4 & 1 & 0 \\ 0 & 0 & 7 & 5 & 1 \\ 1 & 0 & 0 & 7 & 4 \end{pmatrix}.$$

The sequence $(1, 1), (1, 2), (2, 2), (2, 3), (3, 3), (3, 4), (4, 4),$ $(4, 5), (5, 5), (5, 1)$ is a stair sequence for A_1 and B_1.

Lemma 3. *Let A et B be two distinct elements of Ω. There exists a stair sequence for A and B.*

Proof. The proof is by construction. Since A and B are distinct, using the constraints on the sums of rows and columns, there exists a pair of indices $u_1 = (i_1, j_1)$ such that $A(i_1, j_1) > B(i_1, j_1)$. Now using the sum constraint on row i_1, there exists j_2 such that $B(i_1, j_2) < A(i_1, j_2)$. Set $u_2 = (i_1, j_2)$. Similarly, using the sum constraint on column j_2, there exists $i_3 \neq i_1$ such that $A(i_3, j_2) > B(i_3, j_2)$. Set $u_3 = (i_3, j_2)$. Similarly, by the constraint on row i_3, there exists j_4 such that $A(i_3, j_4) < B(i_3, j_4)$. At this step, u_1, u_2, u_3, u_4 are pairwise distinct.

If $j_4 = j_1$, then u_1, u_2, u_3, u_4 is a stair sequence for A and B. Otherwise, by the j_4-column constraint, there exists i_5 such that $B(i_5, j_4) > A(i_5, j_4)$. Now, one can continue the construction until the first step r we get either $i_r = i_s$ or $j_r = j_s$ with $s < r$ (this step exists since the set of possible indexes is finite). Note that we consider the smallest s for which this is case.

- If $i_r = i_s$, $s < r$, the sequence $u_1, u_2, \ldots, u_r$ satisfies the conditions 2., 3. and 4. of Definition 2. Moreover both r and s are odd. The sequence $u_s, \ldots, u_r$ satisfies the Conditions 2. to 5. of Definition 2. Since $r > s$ and by construction, $r - s > 4$. If follows that the sequence $u_r, u_{r-1}, \ldots, u_{s+1}$ is a stair sequence for A and B.

- If $j_r = j_s$, then both r and s are even. The sequence $u_{s+1}, \ldots, u_r$ satifies the Conditions 1. to 5. of Definition 2 and is therefore a stair sequence for A and B.

$\square$

Given two $n \times m$ matrices A and B, the distance from A to B, denoted $d(A, B)$, is defined by:

$$d(A, B) = \sum_{i=1}^{n} \sum_{j=1}^{m} |A(i, j) - B(i, j)|.$$

Lemma 4. *Let A et B be two distinct elements of Ω. There exists $C \in \Omega$ such that $d(C, B) < d(A, B)$ and tuples $t_1, \ldots, t_k$ such that $C = f(\ldots f(f(A, t_1), t_2) \ldots, t_k)$ and for every $\ell \leq k$, $f(\ldots f(f(A, t_1), t_2) \ldots, t_\ell) \in \Omega$.*

Proof. By Lemma 3, there exists a stair sequence $u_1, \ldots, u_r$ for A and B. Without loss of generality (using a permutation of rows and columns) we may assume that $u_{2k+1} = (k, k)$ and $u_{2k} = (k, k+1)$, for $k < \frac{r}{2}$ and $u_r = (\frac{r}{2}, 1)$.

To illustrate the proof, we introduce some $\frac{r}{2} \times \frac{r}{2}$ matrix M over $\{+, -, \min, \max\}$, called *difference matrices*, such that: if $M(i, j) = +$, then $A(i, j) > B(i, j)$; if $M(i, j) = -$, then $A(i, j) < B(i, j)$; if $M(i, j) = \min$, then $A(i, j) = A_{\min}(i, j)$; and if $M(i, j) = \max$, then $A(i, j) = B_{\max}(i, j)$.

Considering for instance the matrices A_1 and B_1 defined before, with a global minimum equal to 0 and global maximum equal to 7, a difference matrix is

$$\begin{pmatrix} + & - & \min & \min & \max \\ \max & + & - & \min & \min \\ \min & \max & + & - & \min \\ \min & \min & \max & + & - \\ - & \min & \min & \max & + \end{pmatrix}.$$

Note that it may exist several difference matrices since, for instance, some $+$ might be replaced by a max

The proof investigates several cases:

Case 0: If $r = 4$, then $k = 1$ and $t_1 = (2, 1, 1, 2)$ works. Indeed, since $B_{\max}(i, j) \geq A(1, 1) > B(1, 1) \geq A_{\min}(i, j)$, one has $A_{\min}(1, 1) \leq A(1, 1) - 1 \leq B_{\max}(1, 1)$. Similarly, $A_{\min}(2, 1) \leq A(2, 1) + 1 \leq B_{\max}(2, 1)$, $A_{\min}(1, 2) \leq A(1, 2) + 1 \leq B_{\max}(1, 2)$ and $A_{\min}(2, 2) \leq A(2, 2) - 1 \leq B_{\max}(2, 2)$. It follows that $C = f(A, (2, 1, 1, 2)) \in \Omega$ and $d(C, B) = d(A, B) - 4 < d(A, B)$. In this case, the following matrix is a difference matrix:

$$\begin{pmatrix} + & - \\ - & + \end{pmatrix}.$$

Case 1: If $r > 4$ and if there exists $3 \leq \ell \leq \frac{r}{2}$ such that $A(\ell - 2, \ell) \neq A_{\min}(\ell - 2, \ell)$, then, as for Case 0, $k = 1$ works with $t_1 = (\ell - 2, \ell - 1, \ell - 1, \ell)$: $f(A, t_1) \in \Omega$. Moreover $d(f(A, t_1), B) = d(A, B) - 4$ if $A(\ell - 2, \ell) > B(\ell - 2, \ell)$; $d(f(A, t_1), B) = d(A, B) - 2$ otherwise. In this case, the following matrix is a difference matrix:

$$\begin{pmatrix} + & - & & & & & \\ & + & - & A(\ell-2,\ell) & & & \\ & & + & & - & & \\ & & & + & & \ddots & \\ & & & & \ddots & - & \\ & & & & & + & - \\ - & & & & & & + \end{pmatrix}.$$

Case 2: If $r > 4$ and Case 1 does not hold and if there exists $1 \leq \ell \leq \frac{r}{2} - 1$ such that $A(\ell + 1, \ell) < B_{\max}(\ell + 1, \ell)$, then, similarly, $k = 1$ and $t_1 = (\ell, \ell + 1, \ell + 1, \ell)$ works. One has $d(f(A, t_1), B) = d(A, B) - 4$ if $A(\ell + 1, \ell) < B(\ell + 1, \ell)$, and $d(f(A, t_1), B) = d(A, B) - 2$ otherwise. In this case, the following matrix is a difference matrix:

$$\begin{pmatrix} + & & - & & \text{min} & & \\ & + & & - & & \text{min} & \\ A(\ell+1,\ell) & & + & & - & & \text{min} \\ & & & + & & \ddots & \ddots \\ & & & & \ddots & & - & \text{min} \\ & & & & & + & - \\ - & & & & & & + \end{pmatrix}.$$

Case 3: If Cases 0 to 2 do not hold and if $A(1, \frac{r}{2}) \neq B_{\max}(1, \frac{r}{2})$, then, similarly, $k = 1$ and $t_1 = (1, \frac{r}{2}, \frac{r}{2}, 1)$ works. One has $d(f(A, t_1), B) = d(A, B) - 4$ if $A(1, \frac{r}{2}) < B(1, \frac{r}{2})$, and $d(f(A, t_1), B) = d(A, B) - 2$ otherwise. In this case, the following matrix is a difference matrix:

$$\begin{pmatrix} + & & - & & \text{min} & & & & A(1, \frac{r}{2}) \\ \text{max} & & + & & - & & \text{min} & & \\ & & \text{max} & & + & & - & & \text{min} \\ & & & & \text{max} & & + & & \ddots & \ddots \\ & & & & & & \ddots & & \ddots & - & \text{min} \\ & & & & & & & \text{max} & & + & - \\ - & & & & & & & & \text{max} & & + \end{pmatrix}.$$

11

Case 4: If Cases 0 to 3 do not hold. Since $A(\frac{r}{2}, \frac{r}{2}) = B_{\max}(1, \frac{r}{2})$ and $A(1, \frac{r}{2} - 2) = A_{\min}(\frac{r}{2}, \frac{r}{2} - 2)$, $i_0 = \max\{i \mid 1 \le i < \frac{r}{2} - 2 \text{ and } A(i, \frac{r}{2}) \ne A_{\min}(i, \frac{r}{2})\}$ exists. In this case $t_1 = (i_0, i_0 + 1, i_0 + 1, \frac{r}{2}), t_2 = (i_0 + 1, i_0 + 2, i_0 + 2, \frac{r}{2}), \dots, t_{\frac{r}{2} - 2 - i_0} = (\frac{r}{2} - 2, \frac{r}{2} - 1, \frac{r}{2} - 1, \frac{r}{2})$ works. With $C = f(\dots f(f(A, t_1), t_2) \dots, t_{\frac{r}{2} - 2 - i_0})$. One has $d(C, B) = d(A, B) - 2 \times (\frac{r}{2} - i_0 - 1)$ if $A(i_0, \frac{r}{2}) > B(i_0, \frac{r}{2})$, and $d(C, B) = d(A, B) - 2 \times (\frac{r}{2} - i_0 - 2)$ otherwise. Moreover, for every $\ell \le \frac{r}{2} - 2 - i_0$, $f(\dots f(f(A, t_1), t_2) \dots, t_\ell) \in \Omega$. In this case, the following matrix is a difference matrix:

$$
\begin{pmatrix}
+ & - & \min & & & & \max \\
\max & + & - & \min & & & \\
& \max & + & - & \min & & A(i_0, \frac{r}{2}) \\
& & \max & + & \ddots & \ddots & \vdots \\
& & & \ddots & \ddots & - & \min \\
& & & & \max & + & - \\
- & & & & & \max & +
\end{pmatrix}.
$$

$\square$

One can now prove Theorem 1.

Proof. If $A = f(B, (i_0, j_0, i_1, j_1))$, then $B = f(A, (i_1, j_1, i_0, j_0))$, proving that the Markov Chain is symmetric.

Let $A_0 \in \Omega$. We define the sequence $(A_k)_{k \ge 0}$ by $A_{k+1} = f(A_k, (1, 1, 2, 2))$. The sequence $A_k(1, 2)$ is decreasing and positive. Therefore, one can define the smallest index k_0 such that $A_{k_0}(1, 2) = A_{k_0+1}(1, 2)$. By construction, one also has $A_{k_0} = A_{k_0+1}$. It follows that the Markov Chain is aperiodic.

Since d is a distance, irreducibility is a direct consequence of Lemma 4. $\square$

Consider the two matrices A_1 and B_1 defined previously with $B_{\max}$ containing only the value 7. Case 4 of the proof can be applied. One has $t_1 = (1, 2, 2, 5)$ and

$$
f(A_1, t_1) = A_2 = \begin{pmatrix}
3 & 1 & 0 & 0 & 6 \\
7 & 3 & 0 & 0 & 1 \\
0 & 7 & 5 & 0 & 0 \\
0 & 0 & 7 & 6 & 0 \\
0 & 0 & 0 & 7 & 5
\end{pmatrix}.
$$

Next, $t_2 = (2, 3, 3, 5)$ and

$$
f(A_2, t_2) = A_3 = \begin{pmatrix}
3 & 1 & 0 & 0 & 6 \\
7 & 3 & 1 & 0 & 0 \\
0 & 7 & 4 & 0 & 1 \\
0 & 0 & 7 & 6 & 0 \\
0 & 0 & 0 & 7 & 5
\end{pmatrix}.
$$

We have $t_3 = (3, 4, 4, 5)$ and

$$f(A_3, t_3) = A_4 = \begin{pmatrix} 3 & 1 & 0 & 0 & 6 \\ 7 & 3 & 1 & 0 & 0 \\ 0 & 7 & 4 & 1 & 0 \\ 0 & 0 & 7 & 5 & 1 \\ 0 & 0 & 0 & 7 & 5 \end{pmatrix}.$$

Finally, $f(A_4, (5, 1, 1, 5)) = B_1$ (Case 0): there is a path from A_1 to B_1 and, since the chain is symmetric, from B_1 to A_1.

4.3 Rapidly Mixing Chains

The chain $\mathcal{M}$ can be classically modified in order to mix faster: once an element of $\mathcal{K}$ is picked up, rather than changing each element by $+1$ or -1, each one is modified by $+a$ or $-a$, where a is picked uniformly in order to respect the constraints of the matrix. This approach, used for instance in [16], allows moving faster, particularly for large N's.

Moving in $\Omega_{n,m}^N(\overline{\mu}, \overline{\nu})$, from matrix M, while (i_0, j_0, i_1, j_1) has been picked in $\mathcal{K}$, a is uniformly chosen such that $a \leq \min\{M(i_0, j_1), M(i_1, j_0)\}$ in order to keep positive elements in the matrix. It can be generalized for constrained Markov Chains. For instance, in $\Omega_{n,m}^N(\overline{\mu}, \overline{\nu})[\alpha, \beta]$, one has $a \geq 1$ and

$$a \leq \min\{\alpha - M(i_0, j_0), \alpha - M(i_1, j_1),$$
$$M(i_0, j_1) - \beta, M(i_1, j_0) - \beta\}.$$

This approach is used in the experiments described in Sections 5 and 6.

5 Convergence of the Markov Chains

We can now generate a matrix that is uniformly distributed when the Markov Chain is run long enough to reach a stationary distribution. The mixing time $t_{\mathrm{mix}}(\varepsilon)$ of an ergodic Markov Chain is the number of steps required in order to be ε-close to the stationary distribution (for the total variation distance, see [8, Chapter 4]). Computing theoretical bounds on mixing time is a hard theoretical problem. For two rowed contingency tables, it is proved in [16] that $t_{\mathrm{mix}}(\varepsilon)$ is in $O(n^2 \log(\frac{N}{\varepsilon}))$ and conjectured that it is in $\Theta(n^2 \log(\frac{n}{\varepsilon}))$. The results are extended and improved in [22] for a fixed number of rows. As far as we know, there are no known results for the general case. A frequently used approach to tackle the convergence problem (when to stop mixing the chain) consists in using statistical test. Starting from a different point of the state space (ideally well spread in the graph), we perform several random walks and we monitor numerical parameter in order to observe the convergence. For our work, used parameters are defined in Section 5.1. Section 5.2 is dedicated to finding different starting points. Convergence experimental results are given in Section 5.3.

5.1 Measures

We apply a set of measures on the matrix at each step of the Markov process to assess its convergence. At first, these measures heavily depend on the initial matrix. However, they eventually converge to

a stationary distribution as the number of steps increases. In the following, we assume that once they converge, the Markov Chain is close to the stationary distribution.

These measures consist in:

- the cost Coefficient-of-Variation (ratio of standard deviation to mean):
$$\sqrt{\frac{1}{nm} \sum_{i=1}^{n} \sum_{j=1}^{m} \left(M(i,j)/\frac{N}{nm} - 1 \right)^2}$$

- the mean of row Coefficients-of-Variation: $\sum_{i=1}^{n} \frac{\sqrt{\frac{1}{m} \sum_{j=1}^{m} (M(i,j) - \frac{\overline{\mu}(i)}{m})^2}}{n\overline{\mu}(i)}$

- the mean of column Coefficients-of-Variation: $\sum_{j=1}^{m} \frac{\sqrt{\frac{1}{n} \sum_{i=1}^{n} (M(i,j) - \frac{\overline{\nu}(j)}{n})^2}}{m\overline{\nu}(j)}$

- Pearson's χ^2 statistic: $\sum_{i=1}^{n} \frac{(M(i,j) - \overline{\mu}(i)\overline{\nu}(j)/N)^2}{\overline{\mu}(i)\overline{\nu}(j)/N}$

- the mean of row correlations: $\frac{1}{n(n-1)/2} \sum_{i=1}^{n-1} \sum_{i'=i+1}^{n} \rho(M(i,.), M(i',.))$

- the mean of column correlations: $\frac{1}{m(m-1)/2} \sum_{j=1}^{m-1} \sum_{j'=j+1}^{m} \rho(M(.,j), M(.,j'))$

where $\rho(M(i,.), M(i',.))$ (resp. $\rho(M(.,j), M(.,j'))$) denotes the Pearson coefficient of correlation between rows i and i' (resp. columns j and j'). When a row or column contains identical values, the related correlations are undefined. When a row (resp. column) sum is zero, the mean of row (resp. column) CV and the χ^2 are undefined.

The first measure is an indicator of the overall variance of the costs. The second two measures indicate whether this variance is distributed on the rows (task heterogeneity) or the columns (machine heterogeneity). The χ^2 is used to assess the proportionality of the costs and the correlations show whether rows or columns are proportional.

5.2 Initial Matrix

The Markov Chain described in Section 4 requires an initial matrix. Before reaching the stationary distribution, the Markov Chain iterates on matrices with similar characteristics to the initial one. However, after enough steps, the Markov Chain eventually converges. We are interested in generating several initial matrices with different characteristics to assess this number of steps. Formally, given $\overline{\mu}$, $\overline{\nu}$, $A_{\min}$ and $B_{\max}$, how to find an element of $\Omega_{n,m}^{N}(\overline{\mu}, \overline{\nu})[A_{\min}, B_{\max}]$ to start the Markov Chain?

We identify three different kinds of matrices for which we propose simple generation methods:

- a *homogeneous* matrix with smallest cost CV (Algorithm 2)

- a *heterogeneous* matrix with largest cost CV (Algorithm 3)

- a *proportional* matrix with smallest Pearson's χ^2 statistic (Algorithm 4)

Ideally, initial matrices could be generated with an exact method (e.g. with an integer programming solver). However, the optimality is not critical to assess the time to converge and Algorithms 2 to 4 have low costs but are not guaranteed.

Moreover, the convergence may be the longest when the search space is the largest, which occurs when the space is the least constrained. Thus, Algorithms 2 to 4 are used to study convergence

---
Algorithm 2: Homogeneous Matrices

Input: Integer vectors $\overline{\mu}$, $\overline{\nu}$

Output: $M \in \Omega_{n,m}^N(\overline{\mu}, \overline{\nu})$

1 begin

2 $M \leftarrow \{0\}_{1 \leq i \leq n, 1 \leq j \leq m}$

3   **while** $\sum_{i=1}^{n} \sum_{j=1}^{n} M(i,j) \neq N$ **do**

4   **if** $\max(\overline{\mu}(i))/m \geq \max(\overline{\nu}(j))/n$ **then**

5 $i \leftarrow \arg\max_i \overline{\mu}(i)$

6   sort $j_1, \ldots, j_m$ such that $\overline{\nu}(j_k) \leq \overline{\nu}(j_{k+1})$

7   **for** $j_k \in \{j_1 \ldots, j_m\}$ **do**

8 $d \leftarrow \min(\overline{\nu}(j_k), \frac{\overline{\mu}(i)}{m-k+1})$

9 $M(i, j_k) \leftarrow M(i, j_k) + d$

10 $\overline{\mu}(i) \leftarrow \overline{\mu}(i) - d$

11 $\overline{\nu}(j_k) \leftarrow \overline{\nu}(j_k) - d$

12   **else**

13   perform the same operation on the transpose matrix (swapping $\overline{\mu}$ and $\overline{\nu}$)

14   **return** M

Algorithm 3: Heterogeneous Matrices

Input: Integer vectors $\overline{\mu}$, $\overline{\nu}$

Output: $M \in \Omega_{n,m}^N(\overline{\mu}, \overline{\nu})$

1 begin

2 $M \leftarrow \{0\}_{1 \leq i \leq n, 1 \leq j \leq m}$

3   **while** $\sum_{i=1}^{n} \sum_{j=1}^{n} M(i,j) \neq N$ **do**

4 $D \leftarrow \min(\overline{\mu}^T \cdot \mathbb{1}_m, \mathbb{1}_n^T \cdot \overline{\nu})$

5 $i_{\max}, j_{\max} \leftarrow \arg\max_{i,j} D(i,j)$

6 $d \leftarrow D(i_{\max}, j_{\max})$

7 $M(i_{\max}, j_{\max}) \leftarrow d$

8 $\overline{\mu}(i_{\max}) \leftarrow \overline{\mu}(i_{\max}) - d$

9 $\overline{\nu}(j_{\max}) \leftarrow \overline{\nu}(j_{\max}) - d$

10   **return** M

Algorithm 4: Proportional Matrices

Input: Integer vectors $\overline{\mu}$, $\overline{\nu}$, integer matrices $A_{\min}$, $B_{\max}$

Output: $M \in \Omega_{n,m}^N(\overline{\mu}, \overline{\nu})[A_{\min}, B_{\max}]$

1 begin

2 $M \leftarrow \max(A_{\min}, \min(\lfloor \overline{\mu}^T \times \overline{\nu}/N + 1/2 \rfloor, B_{\max}))$

3 $\overline{\mu}(i) \leftarrow \overline{\mu}(i) - \sum_{j=1}^m M(i,j)$ for $1 \le i \le n$

4 $\overline{\nu}(j) \leftarrow \overline{\nu}(j) - \sum_{i=1}^n M(i,j)$ for $1 \le j \le m$

5   **while** $\sum_{j=1}^m M(i,j) \ne \overline{\mu}(i)$ *or* $\sum_{i=1}^n M(i,j) \ne \overline{\nu}(j)$ **do**

6   choose random i and j

7 $d \leftarrow 0$

8   **if** $M(i,j) < B_{\max}(i,j), (\overline{\mu}(i) > 0$ or $\overline{\nu}(j) > 0)$ **then** $d \leftarrow 1$

9   **if** $M(i,j) > A_{\min}(i,j), (\overline{\mu}(i) < 0$ or $\overline{\nu}(j) < 0)$ **then** $d \leftarrow -1$

10 $M(i,j) \leftarrow M(i,j) + d$

11 $\overline{\mu}(i) \leftarrow \overline{\mu}(i) - d$

12 $\overline{\nu}(j) \leftarrow \overline{\nu}(j) - d$

13   **return** M

---

without constraints $A_{\min}$ and $B_{\max}$. Only Algorithm 4 supports such constraints and is used to study their effects in Section 6.

Algorithm 2 starts with an empty matrix. Then, it iteratively selects the row (or column) with largest remaining sum. Each element of the row (or column) is assigned to the highest average value. This avoids large elements in the matrix and leads to low variance. Algorithm 3 also starts with an empty matrix. Then, it iteratively assigns the element that can be assigned to the largest possible value. This leads to a few large elements in the final matrix. Algorithm 4 starts with the rounding of the rational proportional matrix (i.e. the matrix in which costs are proportional to the corresponding row and column costs) and proceeds to few random transformations to meet the constraints.

In Algorithms 2 and 3, the argmin and argmax can return any index arbitrarily in case of several minimums. In Algorithm 3, $\mathbb{1}_n$ denotes a vector of n ones. Finally, in Algorithms 3 and 4, $\overline{\mu}^T$ denotes the transpose of $\overline{\mu}$, which is a column vector.

5.3 Experiments

We first illustrate the approach with the example of a 20×10 matrix with $N = 4\,000$ with given $\overline{\mu}$ and $\overline{\nu}$. Starting from three different matrices as defined in Section 5.2, we monitor the measures defined in Section 5.1 in order to observe the convergence (here, approximately after $6\,000$ iterations). It is, for instance, depicted in Figure 5 for the cost CV (diagrams for other measures are similar and seems to converge faster). Next, for every measure, many walks with different $\overline{\mu}$ and $\overline{\nu}$ (but same N) are performed and the value of the measures is reported in boxplots[5] for several walking steps, as in Figure 6 for the CV, allowing to improve the confidence in the hypothesis of convergence. One can observe that the three boxplots are synchronized after about $6\,000$ iterations.

---

[5]Each boxplot consists of a bold line for the median, a box for the quartiles, whiskers that extends to 1.5 times the interquartile range from the box and additional points for outliers.
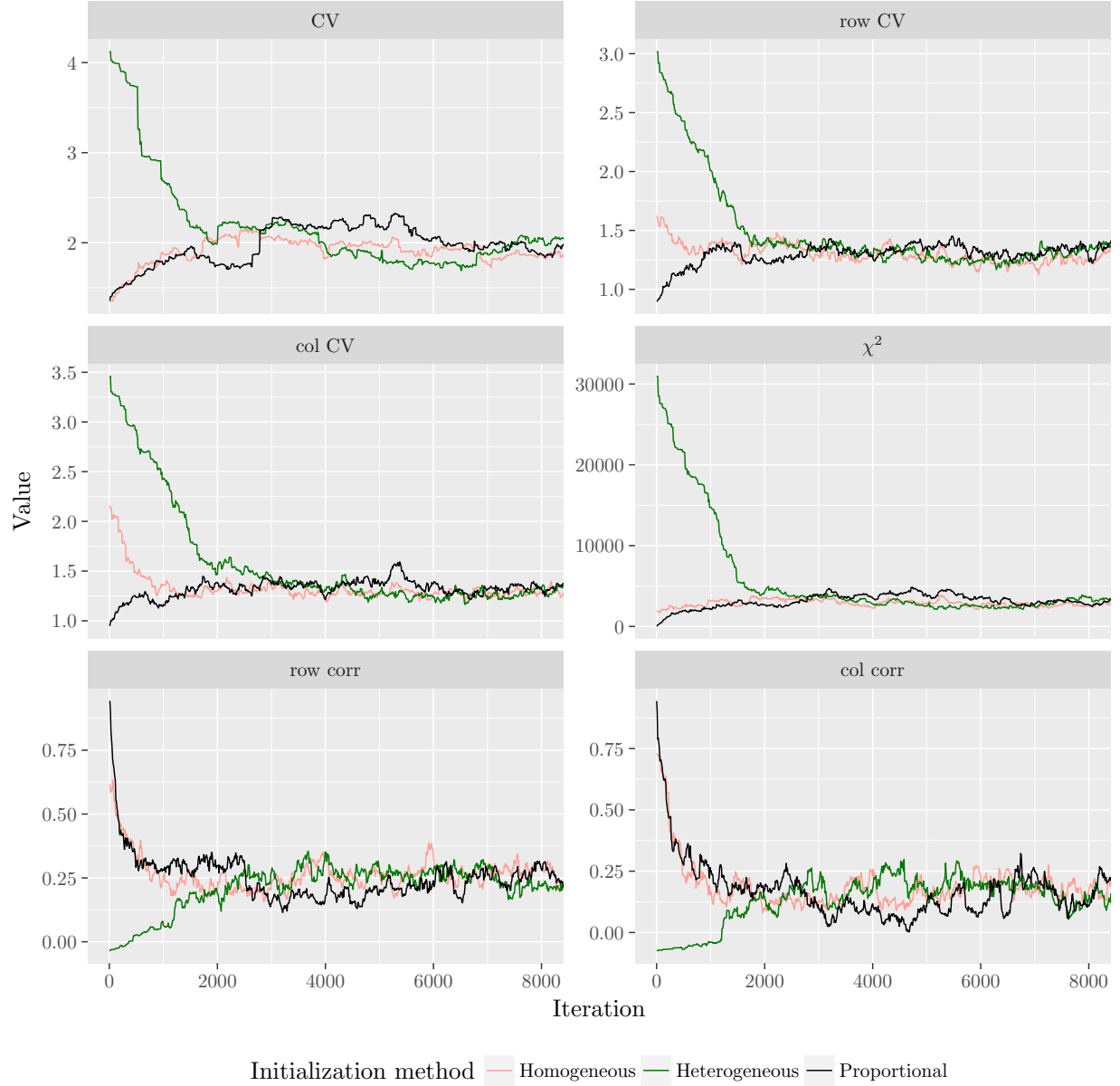
Figure 5: Evolution of the measures for a 20×10 matrix, with $N = 20 \times n \times m = 4\,000$. Initial row and column sums ($\overline{\mu}$ and $\overline{\nu}$) are generated with Algorithm 1 without constraints (i.e. $\alpha = 0$ and $\beta = N$). Initial matrices are generated with Algorithms 2 to 4 without constraints.
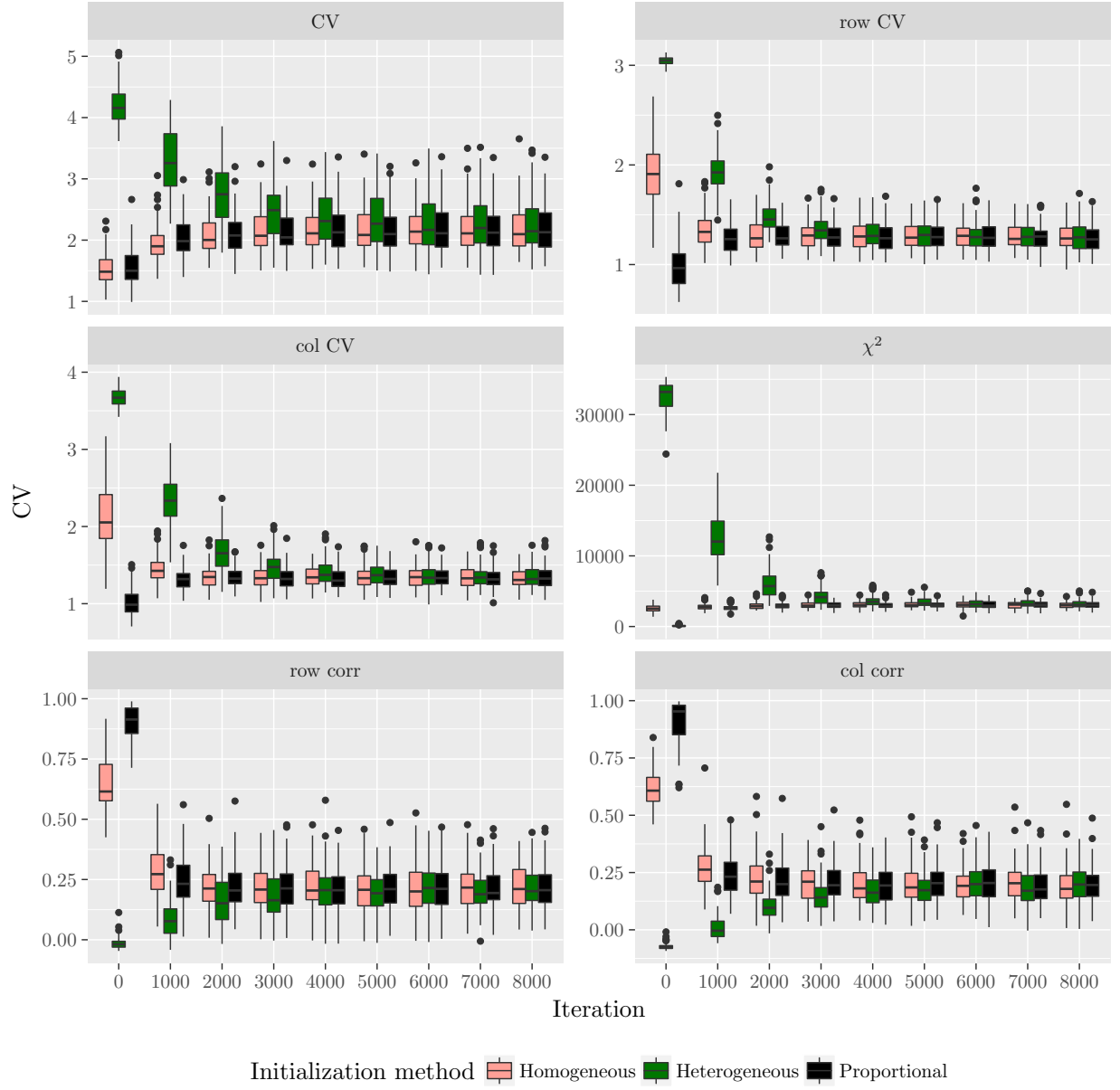
Figure 6: Evolution of the measures for matrices with the same characteristics as in Figure 5. Each boxplot corresponds to 100 matrices, each based on distinct row and column sums. When a row or column sum is zero, all undefined measures are discarded.

(n, m)	mixing time
$(5, 5)$	200
$(5, 10)$	600
$(5, 15)$	1 000
$(10, 10)$	2 500
$(10, 15)$	3 500
$(10, 20)$	6 000
$(25, 10)$	7 500
$(15, 20)$	8 000
$(15, 25)$	13 000
$(20, 25)$	30 000
$(20, 30)$	50 000
$(40, 20)$	65 000
$(40, 40)$	210 000

Table 1: Estimated mixing times with a visual method and with varying number of rows n and columns m.

These experiments have been performed for several matrices sizes, several $\overline{\mu}$, $\overline{\nu}$ generations (with different min/max constraints), and different N. The experimental results seem to point out that the convergence speed is independent of N (assuming that N is large enough to avoid bottleneck issues) and independent of the min/max constraints on $\overline{\mu}$ and $\overline{\nu}$. Estimated convergence time (iteration steps) obtained manually with a visual method (stability for the measures) for several sizes of matrices are reported in Table 1. Experimentally, the mixing (convergence) time seems to be linearly bounded by $nm \log^3(nm)$.

6 Performance Evaluation of Scheduling Algorithms

This section studies the effect of the constraints on the matrix properties (Section 6.1) and on the performance of some scheduling heuristics from the literature (Section 6.2).

This section relies on matrices of size 20×10 with non-zero cost. This is achieved by using $\alpha \geq m$ for $\overline{\mu}$, $\alpha \geq n$ for $\overline{\nu}$ and a matrix $A_{\min}$ containing only ones.

Section 5 provides estimation for the convergence time of the Markov Chain depending on the size of the cost matrix in the absence of constraints on the vectors (α and β) and on the matrix ($A_{\min}$ and $B_{\max}$). We assume that the convergence time does not strongly depend on the constraints. Moreover, this section relies on an inflated number of iterations, i.e. 50 000, for safety, starting from the proportional matrix (Algorithm 4).

6.1 Constraints Effect on Cost Matrix Properties

We want to estimate how the constraints on the $\overline{\mu}$ and $\overline{\nu}$ random generation influence the matrix properties. Figure 7 reports the results. Each row is dedicated to a property from the CV to the column correlation that are presented in Section 5.1, with the inclusion of the $\overline{\mu}$ and $\overline{\nu}$ CV. On the left of the plot, only $\overline{\nu}$ is constrained. In the center only $\overline{\mu}$ and in the right, both $\overline{\mu}$ and $\overline{\nu}$. Constraints are parametrized by a coefficient in $\lambda \in \{0, 0.2, \dots, 1\}$: intuitively, large values of λ

impose strong constraints and limit the CV. The influence of λ on the CV of $\overline{\mu}$ and/or $\overline{\nu}$ is consistent with Figure 3 in Section 3: the value decreases from about 20 to 0 as the constraint increases.

The heterogeneity of a cost matrix can be defined in two ways [5]: using either the CV of $\overline{\mu}$ and $\overline{\nu}$, or using the mean row and column CV. Although constraining $\overline{\mu}$ and $\overline{\nu}$ limits the former kind of heterogeneity, the latter only decreases marginally. To limit the heterogeneity according to both definitions, it is necessary to constraint the matrix with $A_{\min}$ and $B_{\max}$. Figure 8 shows the effect of these additional constraints when the cost matrix cannot deviate too much from an ideal fractional proportional matrix. In particular, $\overline{\mu}$ (resp. $\overline{\nu}$) is constrained with a parameter λ_r (resp. λ_c) as before. The constraint on the matrix is performed with the maximum λ of these two parameters. This idea is to ensure the matrix is similar to a proportional matrix M with $M(i,j) = \frac{\overline{\mu}(i) \times \overline{\nu}(j)}{N}$ when any constraint on the row or column sum vectors is large.

The figure shows that the cost CV decreases as both λ_r and λ_c increase. Moreover, as for the $\overline{\mu}$ (resp. $\overline{\nu}$) CV, the mean column (resp. row) CV decreases as λ_r (resp. λ_c) increases. We can thus control the row and column heterogeneity with λ_r and λ_c, respectively. Note that when reducing the heterogeneity, row or column correlations tend to increase. In particular, large values for λ_r/λ_c lead to jumps from small correlations when $\lambda_r = \lambda_c$ to large row (resp. column) correlation when $\lambda_r = 1$ (resp. $\lambda_c = 1$).

6.2 Constraints Effect on Scheduling Algorithms

Generating random matrices with parameterized constraints allows the assessment of existing scheduling algorithms in different contexts. In this section, we focus on the impact of cost matrix properties on the performance of three heuristics for the problem denoted $R||C_{max}$. This problem consists in assigning a set of independent tasks to machines such that the *makespan* (i.e. maximum completion time on any machine) is minimized. The cost of any task on any machine is provided by the cost matrix and the completion time on any machine is the sum of the costs of all task assigned to it.

The heuristics we consider constitute a diversified selection among the numerous heuristics that have been proposed for this problem in terms of principle and cost:

- *BalSuff*, an efficient heuristic [5] with unknown complexity that balances each task to minimize the makespan.

- *HLPT*, Heterogeneous-Longest-Processing-Time, iteratively assigns the longest task to the machine with minimum completion time in $O(nm + n\log(n))$ steps. This is a natural extension of LPT [23] and variant of HEFT [24] in which the considered cost for each task is its minimal one.

- *EFT*, Earliest-Finish-Time, (or MinMin) is a classic principle, which iteratively assigns each task by selecting the task that finishes the earliest on any machine. Its time complexity is $O(n^2 m)$.

We selected four scenarios that represent the extremes in terms of parameters, heterogeneity and correlation: $\lambda_r = \lambda_c = 0$ with the most heterogeneity and the least correlation, $\lambda_r = 0, \lambda_c = 1$ with a high task and low machine heterogeneity, $\lambda_r = 1, \lambda_c = 0$ with a low task and high machine heterogeneity, and $\lambda_r = 0.75, \lambda_c = 1$ with low heterogeneity and high correlation (the case $\lambda_r = \lambda_c = 1$ lead to identical costs for which all heuristics perform the same). Table 2 gives the mean properties
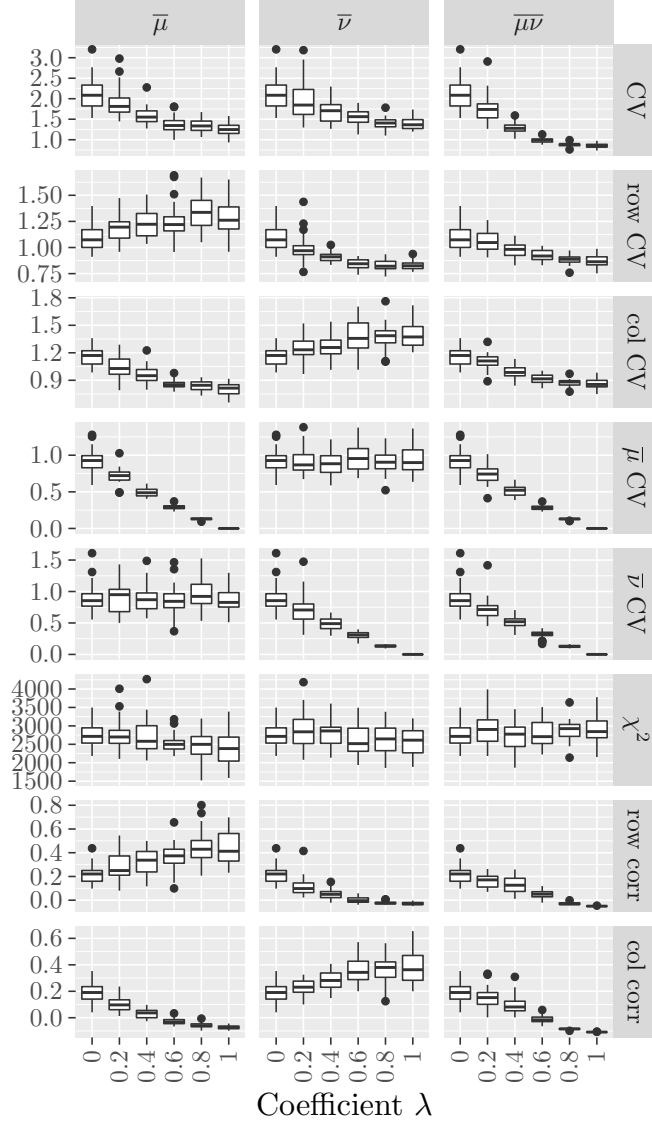
Figure 7: Values for the measures presented in Section 5.1 and the cost sums ($\overline{\mu}$ and $\overline{\nu}$) CV after 50 000 iterations starting with a proportional 20×10 matrix generated with Algorithm 4 with different constraints on $\overline{\mu}$ and/or $\overline{\nu}$ (either ones on the first two columns, both on the third) and with $N = 20 \times n \times m = 4\,000$. The constraint on $\overline{\mu}$ (resp. $\overline{\nu}$) is parameterized by a coefficient $0 \leq \lambda \leq 1$ such that $\alpha = \lfloor \frac{\lambda N}{n} \rfloor$ (resp. $\lfloor \frac{\lambda N}{m} \rfloor$) and $\beta = \lceil \frac{N}{\lambda n} \rceil$ (resp. $\lceil \frac{N}{\lambda m} \rceil$), with the convention $1/0 = +\infty$. Each matrix contains non-zero costs. Each boxplot corresponds to 30 matrices, each based on distinct row and column sums.
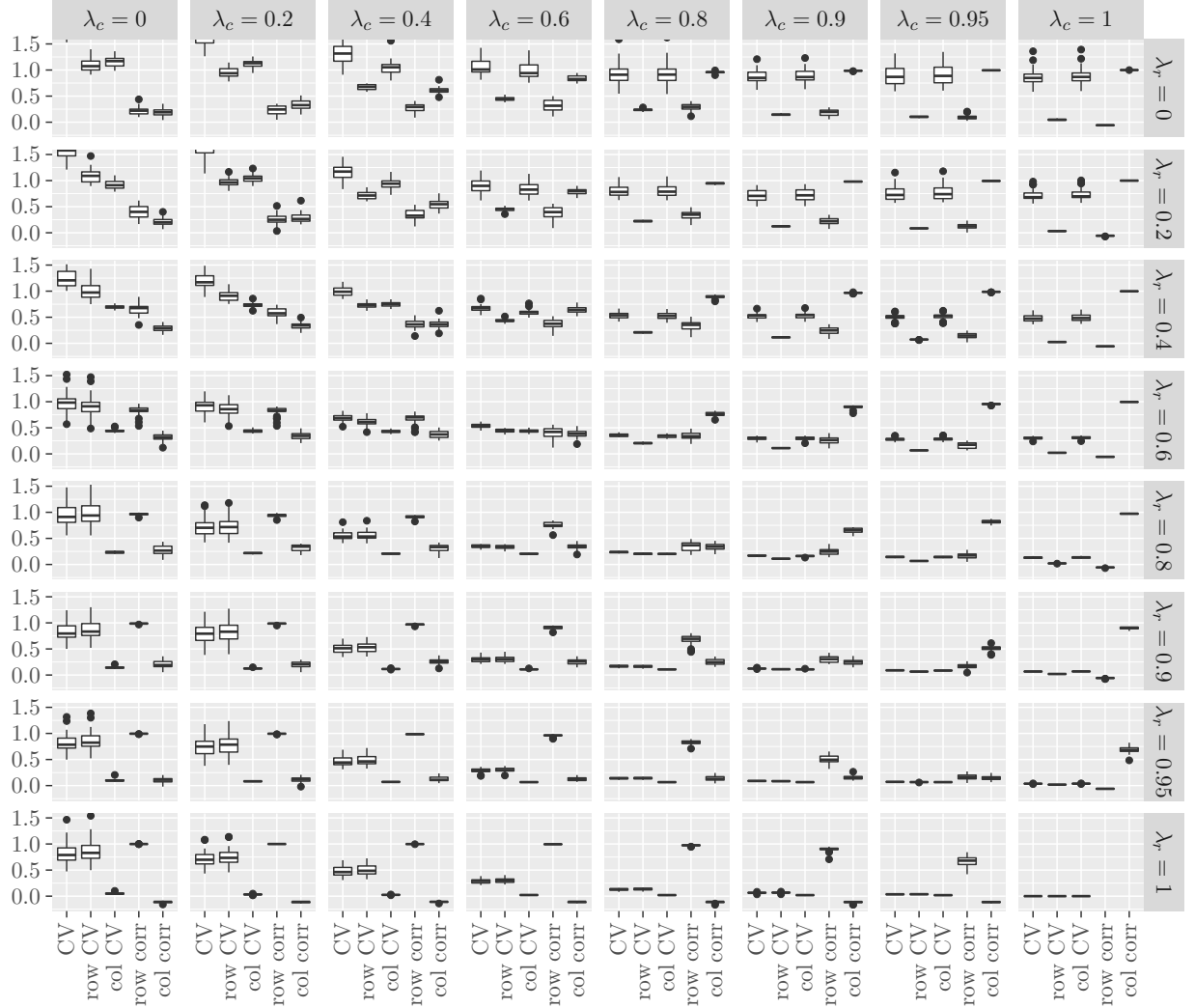
Figure 8: Values for the measures presented in Section 5.1 and the cost sums ($\overline{\mu}$ and $\overline{\nu}$) CV after $50\,000$ iterations starting with a proportional 20×10 matrix generated with Algorithm 4 with different constraints on $\overline{\mu}$, $\overline{\nu}$ and the matrix, and with $N = 20 \times n \times m = 4\,000$. The constraint on $\overline{\mu}$ (resp. $\overline{\nu}$) is parameterized by a coefficient $0 \le \lambda_r \le 1$ (resp. $0 \le \lambda_c \le 1$) such that $\alpha = \lfloor \frac{\lambda_r N}{n} \rfloor$ (resp. $\lfloor \frac{\lambda_c N}{m} \rfloor$) and $\beta = \lceil \frac{N}{\lambda_r n} \rceil$ (resp. $\lceil \frac{N}{\lambda_c m} \rceil$), with the convention $1/0 = +\infty$. The constraint on the matrix is parameterized by a coefficient $\lambda = \max(\lambda_r, \lambda_c)$ such that $A_{\min} = \lfloor \lambda M \rfloor$ and $B_{\max} = \lceil M/\lambda \rceil$ with $M(i, j) = \frac{\overline{\mu}(i) \times \overline{\nu}(j)}{N}$. Each matrix contains non-zero costs. Each boxplot corresponds to 30 matrices, each based on distinct row and column sums. When $\lambda_r = \lambda_c = 1$, all costs are identical and the correlations are discarded.
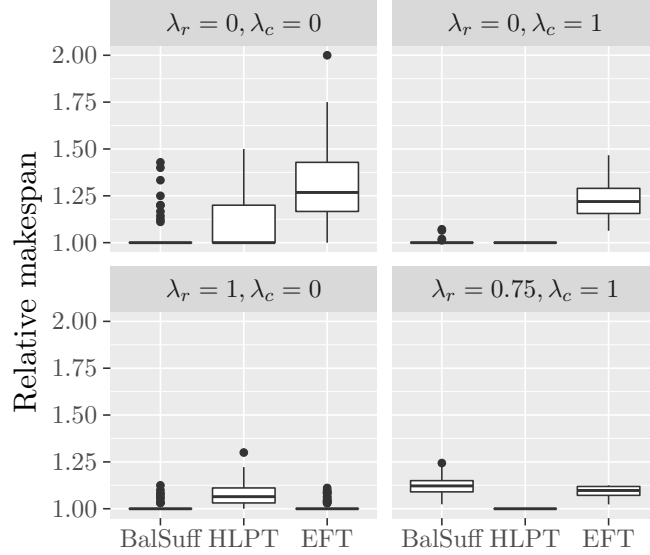
Figure 9: Ratios of makespan to the best among BalSuff, HLPT and EFT. The cost matrices were generated as in Figure 8. Each boxplot corresponds to 100 cost matrices, each based on distinct row and column sums.

λ_r	λ_c	CV	row CV	col CV	$\overline{\mu}$ CV	$\overline{\nu}$ CV	χ^2	row corr	col corr
0	0	2.1	1.1	1.2	0.9	0.87	2831	0.21	0.18
0	1	0.88	0.051	0.9	0.9	0	4.2	-0.058	1
1	0	0.86	0.9	0.051	0	0.9	1.7	1	-0.11
0.75	1	0.17	0.02	0.17	0.17	0	4.3	-0.058	0.98

Table 2: Mean properties over 100 matrices generated as in Figure 9 for each pair of parameters λ_r and λ_c.

for each scenario with 100 matrices each. Figure 9 depicts the results: for each scenario and matrix, the makespan for each heuristic was divided by the best one among the three. All heuristics exhibit different behaviors that depends on the scenario. BalSuff outperforms its competitors except when $\lambda_r = 0.75$ and $\lambda_c = 1$, in which case it is even the worst. HLPT is always the best when $\lambda_c = 1$. In this case, each task has similar costs on any machine. This corresponds to the problem $P||C_{\max}$, for which LPT, the algorithm from which is inspired HLPT, was proposed with an approximation ratio of 4/3 [23]. The near-optimality of HLPT for instances with large row and low column heterogeneity is consistent with the literature [5]. Finally, EFT performs poorly except when $\lambda_r = 1$ and $\lambda_c = 0$. In this case, tasks are identical and it relates to the problem $Q|p_i = 1|C_{\max}$. These instances, for which the row correlation is high and column correlation is low, have been shown to be the easiest for EFT [6].

7 Conclusion

Random instance generation allows broader experimental campaigns but can be hindered by bias in the absence of guarantee on the distribution of the instances. This work focuses on the generation of cost matrices, which can be used in a wide range of scheduling problems to assess the performance of any proposed solution. We propose a Markov Chain Monte Carlo approach to draw random cost matrices from a uniform distribution: at each iteration, some costs in the matrix are shuffled such that the sum of the costs on each row and column remains unchanged. By proving its ergodicity and symmetry, we ensure that its stationary distribution is uniform over the set of feasible instances. Moreover, the result holds when restricting the set of feasible instances to limit their heterogeneity. Finally, experiments were consistent with previous studies in the literature. Although constraining the matrix generation with a minimum and maximum matrices leads to large correlations, it remains to determine the drawbacks of this approach and whether there could be more relevant solutions (such as using a simulated annealing to increase the correlations starting from a uncorrelated ones). A more prospective future direction would be to apply the current methodology on the generation of other types of instances such as task graphs.

Acknowledgments

References

[1] R. Fagin, "Probabilities on finite models," *J. Symb. Log.*, vol. 41, no. 1, pp. 50–58, 1976.

[2] L.-C. Canon and L. Philippe, "On the Heterogeneity Bias of Cost Matrices when Assessing Scheduling Algorithms," FEMTO-ST, Tech. Rep. RR-FEMTO-ST-8663, Mar. 2015.

[3] H. Arabnejad and J. G. Barbosa, "List scheduling algorithm for heterogeneous systems by an optimistic cost table," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 3, pp. 682–694, 2014.

[4] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. R. Kan, "Optimization and approximation in deterministic sequencing and scheduling: a survey," *Annals of Discrete Mathematics*, vol. 5, pp. 287–326, 1979.

[5] L.-C. Canon and L. Philippe, "On the heterogeneity bias of cost matrices for assessing scheduling algorithms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 6, pp. 1675–1688, 2017.

[6] L.-C. Canon, P.-C. Héam, and L. Philippe, "Controlling the correlation of cost matrices to assess scheduling algorithm performance on heterogeneous platforms," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 15, 2017.

[7] P. Flajolet, P. Zimmermann, and B. V. Cutsem, "A calculus for the random generation of labelled combinatorial structures," *Theor. Comput. Sci.*, vol. 132, no. 2, pp. 1–35, 1994.

[8] D. A. Levin, Y. Peres, and E. L. Wilmer, *Markov chains and mixing times.* American Mathematical Society, 2006.

[9] L.-C. Canon, P.-C. Héam, and M. El Sayah, "Code to assess the properties of the cost matrices generated using mcmc," Mar. 2018. [Online]. Available: https://figshare.com/articles/Code_to_assess_the_properties_of_the_cost_matrices_generated_using_MCMC/6011660/1

[10] S. Ali, H. J. Siegel, M. Maheswaran, D. Hensgen, and S. Ali, "Representing task and machine heterogeneities for heterogeneous computing systems," *Tamkang Journal of Science and Engineering*, vol. 3, no. 3, pp. 195–208, 2000.

[11] S. Ali, H. J. Siegel, M. Maheswaran, and D. Hensgen, "Task execution time modeling for heterogeneous computing systems," in *Heterogeneous Computing Workshop (HCW).* IEEE, 2000, pp. 185–199.

[12] K. Pearson, "On the theory of contengency and its relation to association and normal correlation," *Drapers' Company Reserach Memoirs*, 1904.

[13] P. Diaconis and L. S. Coste, "Random walk on contingency tables with mixed row and column sums," Harvard University, Department of Mathematics, Tech. Rep., 1995.

[14] M. del Carmen Pardo, "On testing indenpendence in multidimensional contingency tables with stratified random sampling," *Inf. Sci.*, vol. 78, no. 1-2, pp. 101–118, 1994. [Online]. Available: https://doi.org/10.1016/0020-0255(94)90022-1

[15] D. Hernek, "Random generation of 2×n contingency tables," *Random Struct. Algorithms*, vol. 13, no. 1, pp. 71–79, 1998.

[16] M. E. Dyer and C. S. Greenhill, "Polynomial-time counting and sampling of two-rowed contingency tables," *Theor. Comput. Sci.*, vol. 246, no. 1-2, pp. 265–278, 2000. [Online]. Available: https://doi.org/10.1016/S0304-3975(99)00136-X

[17] S. DeSalvo and J. Y. Zhao, "Random sampling of contingency tables via probabilistic divide-and-conquer," *CoRR*, 2015.

[18] D. I. G. Amalarethinam and P. Muthulakshmi, "Dagitizer – a tool to generate directed acyclic graph through randomizer to model scheduling in grid computing," in *Advances in Computer Science, Engineering & Applications*, D. C. Wyld, J. Zizka, and D. Nagamalai, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 969–978.

[19] D. Cordeiro, G. Mounié, S. Perarnau, D. Trystram, J. Vincent, and F. Wagner, "Random graph generation for scheduling simulations," in *3rd International Conference on Simulation Tools and Techniques, SIMUTools '10, Malaga, Spain - March 16 - 18, 2010.* ICST/ACM, 2010, p. 60.

[20] A. Denise and P. Zimmermann, "Uniform random generation of decomposable structures using floating-point arithmetic," *Theor. Comput. Sci.*, vol. 218, no. 2, pp. 233–248, 1999.

[21] P. Duchon, P. Flajolet, G. Louchard, and G. Schaeffer, "Boltzmann samplers for the random generation of combinatorial structures," *Combinatorics, Probability & Computing*, vol. 13, no. 4-5, pp. 577–625, 2004. [Online]. Available: https://doi.org/10.1017/S0963548304006315

[22] M. Cryan, M. Dyer, L. A. Goldberg, M. Jerrum, and R. Martin, "Rapidly mixing markov chains for sampling contingency tables with a constant number of rows," *SIAM Journal on Computing*, vol. 36, pp. 247–278, 2006.

[23] R. L. Graham, "Bounds on Multiprocessing Timing Anomalies," *Journal of Applied Mathematics*, vol. 17, no. 2, pp. 416–429, 1969.

[24] H. Topcuoglu, S. Hariri, and M.-y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260–274, 2002.

A   Notation

Table 3 provides a list of the most used notations in this report.

Symbol	Definition
n	Number of rows (tasks)
m	Number of columns (machines)
$M(i,j)$	Element on the ith row and jth column of matrix M
N	Sum of elements in a matrix ($\sum_{i,j} M(i,j)$)
$\overline{\mu}$	Vector of size n. $\frac{\overline{\mu}(i)}{m}$ is the mean cost of the ith task.
$\overline{\nu}$	Vector of size m. $\frac{\overline{\nu}(j)}{n}$ is the mean cost on the jth machine.
$H_{N,n}^{\alpha,\beta}$	Elements $\overline{v} \in \mathbb{N}^n$ s.t. $\alpha \le \overline{v}(i) \le \beta$ and $\sum_{i=1}^{n} \overline{v}(i) = N$.
$h_{N,n}^{\alpha,\beta}$	Cardinal of $H_{N,n}^{\alpha,\beta}$.
$d(A,B)$	Distance between matrices A and B.
$\Omega_{n,m}^{N}(\overline{\mu},\overline{\nu})$	Set of contingency tables of sum N and sums of rows and columns $\overline{\mu}$ and $\overline{\nu}$.
$\alpha,\ \beta$	Scalar constraints on minimal/maximal values for generated matrices.
$\overline{\alpha},\ \overline{\beta}$	Vector constraints on minimal/maximal values for generated matrices.
$A_{\min},\ B_{\max}$	Matrix constraints on minimal/maximal values for generated matrices.
$\Omega_{n,m}^{N}(\overline{\mu},\overline{\nu})[...]$	Subset of $\Omega_{n,m}^{N}(\overline{\mu},\overline{\nu})$ min/max-constrained by [...].
$f(\cdot,\cdot)$	Random mapping for the Markov Chains.

Table 3: List of notations.