Random Generation for the Performance Evaluation of Scheduling Algorithms

Mohamad El Sayah

THÈSE DE DOCTORAT DE L'ÉTABLISSEMENT UNIVERSITÉ BOURGOGNE – FRANCHE-COMTÉ

PRÉPARÉE À L'UNIVERSITÉ DE FRANCHE-COMTÉ

Ecole doctorale n°37

Sciences pour l'Ingénieur et Microtechniques

Doctorat d'Informatique

Par

Mohamad EL SAYAH

Random Generation for the Performance Evaluation of Scheduling Algorithms

Thèse présentée et soutenue à Besançon, le 20/11/2019

Composition du Jury :

NICOD Jean-Marc	Professeur à l'ENSMM Besançon	Président
BEAUMONT Olivier	Directeur de recherche à Inria Bordeaux - Sud-Ouest	Rapporteur
LEGRAND Arnaud	Directeur de recherche à CNRS Grenoble	Rapporteur
DUFOSSÉ Fanny	Chargée de recherche à Inria Grenoble	Examinatrice
CANON Louis-Claude	Maître de Conférences à l'Université de Franche-Comté	Co-directeur de thèse
HÉAM Pierre-Cyrille	Professeur à l'Université de Franche-Comté	Directeur de thèse

école doctorale sciences pour l'ingénieur et microtechniques

Titre : Génération Aléatoire pour l'Évaluation de Performance d'Algorithmes d'Ordonnancement

Mots clés : ordonnancement, génération aléatoire, instances, DAGs, matrices de coûts, uniforme

Résumé : L'objectif du travail réalisé dans cette thèse consiste à mettre au point des techniques pour l'évaluation de la performance d'algorithmes dans le contexte de l'ordonnancement. Dans de nombreuses branches de la recherche en informatique, de très nombreuses publications mettent en avant de nouveaux algorithmes évalués comme plus performants que les précédents; mais le protocole expérimental utilisé est souvent mal ou trop peu décrit, voire biaisé, ne permettant ni une reproductibilité des résultats, ni de donner une confiance suffisante sur l'ampleur de l'amélioration. Cela suppose d'analyser finement les problèmes considérés afin de définir ce que peuvent être des instances pertinentes.

Dans cette thèse nous étudions deux types d'instances, les DAGs et les matrices de coûts. Pour chaque type nous étudions un problème d'ordonnancement différent. Nous étudions des propriétés qui ont tendances à influencer la structure des instances générées aléatoirement, puis analysons des méthodes existantes de génération aléatoire vis-à-vis des propriétés séléctionnées. Ensuite, nous proposons un générateur aléatoire capable de générer de façon uniforme des matrices de coûts tout en controlant leurs hétérogénéités, puis nous analysons l'impact des générateurs des DAGs et des matrices de coûts sur la performance de certaines heuristiques d'ordonnancement.

Title : Random Generation for the Performance Evaluation of Scheduling Algorithms

Keywords : scheduling, random generation, instances, DAGs, cost matrices, uniform

Abstract : The objective of the work done in this thesis is to develop techniques for evaluating the performance of algorithms in the context of scheduling. In many branches of computer science, numerous publications highlight new algorithms evaluated as more efficient than the previous ones; but the experimental protocol used is often bad or poorly described, or even biased, allowing neither a reproducibility of the results nor giving sufficient confidence on the extent of the improvement. This entails finely analyzing the problems considered in order to define what may be relevant instances.

In this thesis we study two types of instances, DAGs and cost matrices. For each type we study a different scheduling problem. We study properties that tend to influence the structure of randomly generated instances, then analyze existing random generation methods for the selected properties. Then, we propose a random generator able to generate cost matrices in a uniform way while controlling their heterogeneities, then we analyze the impact of the generators of DAGs and cost matrices on the performance of some scheduling heuristics.

UBFC UNIVERSITE BOURGOGNE FRANCHE-COMTE

Université Bourgogne Franche-Comté
32, avenue de l'Observatoire
25000 Besançon

Acknowledgement

I express here my gratitude to everyone who directly or indirectly helped me during the course of my PhD.

First and foremost, I would like to express my deep and sincere gratitude to my supervisors **Dr. Pierre-Cyrille Héam** and **Dr. Louis-Claude Canon** for the continuous support of my PhD research, for their patience, enthusiasm, and immense knowledge. Their dynamism, vision, sincerity, and motivation have deeply inspired me. They have taught me the methodology to carry out the research and to present the research work clearly as possible. Their guidance helped me in all the time of research and their valuable advices have helped me in shaping this thesis in a better way and making my PhD experience more productive. It was a great privilege and honor to work and study under their guidance. The incredibly valuable sessions spent together shaped my understanding for the subject. Time and again, I felt that either it be my scientific or administrative troubles, once they used to come under their notice, they used to either get sorted or they used to assist me in having the means to do so myself. I would also like to thank them for their friendship, empathy, and great sense of humour. I cannot express in words the amount of reverence I feel for them. I could not have imagined having better advisors and mentors for my PhD study. I will always be indebted to them.

I am thankful to **Dr. Olivier Beaumont** and **Dr. Arnaud Legrand** for reviewing this research work. I extend my sincere appreciation to **Dr. Jean-Marc Nicod** and **Dr. Fanny Dufossé** for acting as examiners. I am honored by their participation in the jury committee.

I am grateful to the **Republic of France**, the country whose welcomed me with open arms, continued support, and facilitation.

My appreciation also goes to the members of **Université de Franche-Comté** and **École doctorale SPIM**. I must also mention all the members of the **DISC laboratory**, who have provided me with a great environment in which to do science.

Special thanks are due to **Dr. Nicolas Janey** for his help with lab courses I gave at the university of Franche-Comté.

I would like to say to **Ayham** that you have been my friend and colleague. Thank you for always being interested in listening to the progress in my work.

I also need to thank my dear colleague **Guillaume** for his help.

I would like to thank my dear PhD students for the good time sepent together, especially **Jean-Philippe** and **Vahana**.

Special thanks are due to **Dr. Francis Rousseaux** for the help and support he had offered me during my studies at Université de Reims Champagne-Ardenne.

Some special words of gratitude go to my friend **Mostafa** who has always been a major source of support when things would get a bit discouraging.

Last, but not the least, I am forever indebted to my greatest parents, my father **Khodr** and my mother **Rola**, my wonderful sisters and teachers **Mayssa** and **Dima**, and my humorist brother **Dr. Zaki**, for always being there, to care, protect and raise me. I am extremely thankful to them for their rightful teachings and unconditional love, prayers, sacrifices, and for all of their support over the years in the many forms it has taken. They always represent an endless source of encouragement that kept my spirits high. Also, I would like to express my deep love to my adorable nieces **Lilia**, **Léna**, and **Tiana**, and to my magnificent brother-in-law **Omar**. You are all my other half. Staying away from you has been the biggest ordeal.

Finally, I dedicate this thesis to my family, all the people who love me, and the soul of my grandfather **Abou Nabil**.

Contents

Part I

Introduction and Preliminaries

Chapter 1

Introduction

Contents

1.1 Context

A scheduling problem involves executing a set of activities on a set of available resources while following some rules to provide the maximum possible effectiveness. Scheduling problems are found in several fields. For instance, we can find scheduling problems in transportation, computer science, telecommunication, management of projects, ...

In this thesis, we are interested in the evaluation of the performance of heuristics in the context of scheduling. Since our aim is to evaluate scheduling heuristics, the natural question to ask is: we have a lot of heuristics, which one is the best? Practically, several tools exist to make an evaluation of the performance of scheduling heuristics. For instance, we can evaluate the performance of scheduling heuristics using data sets that correspond to real applications, or using mathematical analytical methods, or using an exhaustive generation, ... When one wishes to

study the properties of a combinatorial object, the ideal would be to make an experimental study with the help of an exhaustive generator. Effectively, when the search space is too big, the exhaustive generator will not work. Due to this issue, in this thesis we do a random generation of instances to sample the search space.

However, random generation methods are prone to bias when they rely on random instances with an uncontrolled or irrelevant distribution. It is thus crucial to provide guarantees on the random distribution of generated instances by ensuring a known selection of any instance among all possible ones. The empirical assessment is critical to determine the best scheduling heuristics on any parallel platform. Simulation is an effective tool to quantify the quality of scheduling heuristics. It can be performed with a large variety of environments and application models, resulting in broader conclusions.

1.2 Scheduling

In this thesis, we are interested in the analysis of the performance of some scheduling algorithms focusing on the random generation of instances. It is nevertheless essential, before being able to tackle this subject, to recall some basic definitions in scheduling. We will naturally limit ourselves to the part of the field that interests us more particularly, namely the scheduling for parallel machines. First, we will recall the general definition of the problem of scheduling. Then, we will describe the classical notation used in the literature to classify the versions of the latter.

Let us consider a set of machines $M = \{M_1, M_2, \ldots, M_m\}$ and a set of tasks to achieve $T = \{T_1, T_2, \ldots, T_n\}$ where m is the number of machines and n the number of tasks. Informally, we can define a scheduling problem as follow: a scheduling consists of a function that associates a start date to each task and another function that allocates a processor to the task. This scheduling respects a number of constraints, such as the fact that a single task can use a given machine at a specific time. More simply, a scheduling problem is defined as the distribution of a set of tasks to a set of machines, with the objective of optimizing one or more performance criteria.

An instance of a scheduling problem is characterized by the description of tasks, machines and objectives. To simplify this characterization, Graham [GLLK79] proposed the three-field notation: $\alpha|\beta|\gamma$. In this notation, the α field defines the type of machines used, the field β the characteristics of an instance of the problem and γ the objective function to be optimized. We detail here some of the notations,

and therefore problems, encountered in the literature.

For the α, three notations are used: P, Q and R. The symbol P denotes the problems for which identical processors are used to perform the tasks. Each task can therefore be executed on each processor with the same time. The symbol Q denotes uniform heterogeneous processors, i.e. each processor j has a certain processing speed of the instructions s_j and therefore executes a task i containing o_i operations in $p_{ij} = o_i s_j$. Finally, the symbol R denotes non-uniform heterogeneous processors, i.e. the execution time of a task is independent from one processor to another, and is therefore noted p_{ij}. Note also the use of 1 as a symbol for the first field, i.e. that the machine has only one processor. The field β contains the notation p_i for the execution time of a task, especially in the case of unitary tasks: $p_i = 1$, i.e. all tasks take exactly the same unitary time to be executed. The availability dates are r_i and the deadline dates d_i. The *prec* notation indicates the presence of dependencies between the tasks. Finally, the γ field represents the objective functions. Among these functions, the ones relying on the end dates denoted by C_i are the most popular. A classical objective in the literature is the minimization of the maximum C_i, which is called the *makespan* (C_{max}). There is also the minimization of the sum of the completion dates. It is very useful for studying the average completion time of the tasks.

Note that in some models the tasks are not sequential: they may for example require a fixed number of processors. In this thesis we are interested in sequential tasks.

1.3 Motivation and Problem Statement

In order to analyze existing random generators and/or new random generators of instances, the first step is to identify the type of instances. In our research, we focus on two types of instances: directed acyclic graphs (DAGs) and cost matrices. After choosing the types of instances, it remains to specify the main questions to which this thesis respond. Basically, four major questions are asked:

1. As mentioned in Section 1.1, some instances can be straightforward to solve. Therefore, to avoid easy instances, the properties of such instances must be analyzed.

   What are the properties of each type of instances (DAGs and cost matrices)?

2. Next, for both types of instances, it will be important to make an analysis of

existing random generators regarding the identified properties.

How are existing random generators behaving regarding the identified properties?

3. In the literature, uniform random generators of instances exist. Some of existing generators ensure some properties of instances but provide no formal guarantee on the distribution of the instances. Other methods exist, some of them with stronger formal guarantees. We are looking for a method that can generate, uniformly at random, instances that have a given set of properties.

   How to ensure a uniform distribution among instances that have given properties?

4. Generating random instances allows the assessment of existing scheduling algorithms in different contexts.

   How are scheduling algorithms performing depending on the random generator?

1.4 Contributions

The thesis responds to several questions mentioned in Section 1.3. Regarding Part II (which concerns DAGs), first, we identify a list of 34 DAG properties and focus on a selection of 8 such properties. Among these, the *mass* quantifies how much an instance can be decomposed into smaller ones. Second, existing random generation methods are formally analyzed and empirically assessed with respect to the selected properties. We establish the sub-exponential generic time complexity for decomposable scheduling problems with uniform DAGs. Last, we study how the generation methods impact scheduling heuristics with unitary costs for the problem denoted by $P|p_j = 1, prec|C_{max}$.

Regarding Part III (which concerns cost matrices), we focus on the study of two properties of cost matrices: the heterogeneity and the correlation. We show the advantages and the limitations of existing generation methods of cost matrices regarding the selected properties. Then, we propose a Markov Chain Monte Carlo approach to draw random constrained cost matrices from a uniform distribution. Finally, we study how our random cost matrices generator impact scheduling heuristics for the problem denoted by $R||C_{max}$.

Table 1.4 represents a summary of the four main questions in the thesis with our contributions.

Questions	DAGs	Cost Matrices
Properties identification	Identification of 34 and selection of 8 such properties	Summary of properties
Analysis of generation methods regarding properties	Analysis of existing random generation methods regarding properties	Behavior of existing generation methods regarding properties
Uniform random generation (constrained instances)		Recursive method for average costs and MCMC for cost matrices
Algorithms performance evaluation regarding generation methods	Analysis for the problem $P\|p_j = 1, prec\|C_{max}$	Analysis for the problem $R\|\|C_{max}$

Table 1.1: Summary of main questions and contributions

1.5 Thesis outline

The thesis is structured as follow:

- Part I is composed of two chapters. Chapter 1, the current chapter, is dedicated to a general introduction, a summary of the main questions of the thesis and our contributions. In Chapter 2, our interest is about the uniform random generation. First, we explain the uniform random generation of combinatorial objects. Second, we show how to generate uniformly at random instances using a recursive method. Then, we give some definitions of Markov Chain process and we show how to be close to the uniform distribution using this approach. Finally, we compare between both methods of generation by an illustration of a constrained uniform random generation of a vector.
- Part II is composed of three chapters related to the random generation of DAGs. In Chapter 3, we expose the related works of the random generation of DAGs. We list some properties of DAGs and we mention the existing tools to generate DAGs in the context of scheduling in parallel systems and the sets of task graphs. Then, we describe some ad hoc method to generate DAGs and we discuss the uniformity of the random generation of DAGs. Then, we recall the problem statement concerning the random generation of DAGs after defining the studied scheduling problem. In Chapter 4, we analyze properties of 8 special DAGs and we study four existing random generation methods of DAGs according to these properties. Finally, in Chapter 5, we study the impact of the four random generators (mentioned in Chapter 4) on three different scheduling algorithms.

- Part III is composed of three chapters related to the random generation of cost matrices. In Chapter 6, first, we recall the definition of a cost matrix with its properties. Second, we analyze the advantages and the limitations of existing generation methods of cost matrices. Then, we mention contingency tables and we show how to generate uniformly at random cost matrices as contingency tables using MCMC process. Finally, we recall the studied problem in this part that is how to ensure a uniform distribution among a set of cost matrices that have a given task and machine heterogeneity. In Chapter 7, first, we formally prove the uniformity of constrained random generation of cost matrices. Second, we show how can a Markov Chain mix faster. Then, we define three extremely different cost matrices that represent the initial states of three Markov Chains to assess the convergence time. Finally, we analyze the effect of our constraints on the properties of the generated cost matrices. In Chapter 8, we analyze the effect of the constraints (used in the generation of cost matrices) on three different scheduling algorithms.
- Part IV is dedicated to the general conclusions and perspectives.

1.6 Publications

Regarding the study of DAGs, our results were accepted to be presented at the 25th International European Conference on Parallel and Distributed Computing (Euro-Par 2019). Moreover, we published a research report [CSH19] in order to publish, in the future, an extended journal version of our results. Also, the artifacts for our paper were accepted in Euro-Par, i.e. the support material (e.g., source code, tools, models) to assess the reproducibility of our experimental results[1].

Regarding the study of cost matrices, our results were presented at the 16th annual meeting of the International Conference on High Performance Computing and Simulation (HPCS 2018) [CESH18]. In addition, we published a research report [CESH18], which includes more details about our results.

[1] https://doi.org/10.6084/m9.figshare.8397623

Chapter 2

Uniform Random Generation

Contents

2.1 Introduction

In this thesis, we are mostly interested in uniform random generation, i.e. all objects of the same size have the same probability of being randomly generated. Also, we study particular random generators. Yet, for some problems, uniformly generated instances do not often correspond to real cases, thus these instances are

uninteresting. For instance, in uniformly distributed random graphs (using the Erdős-Rényi algorithm), the probability that the diameter is 2 tends exponentially to 1 as the size of the graph tends to infinity [Fag76]. Studying the problem characteristics to constrain the universe of the uniform random generation on a category of instances is thus critical. For instance, we might be interested by the uniform random generation of graphs with diameter equals to ten.

Section 2.1 is dedicated to the introduction of the random generation of combinatorial objects. In Section 2.2, we explain how the recursive random generation works and we illustrate this method by generating binary trees. Then, in Section 2.3, we mention some Markov Chain properties, then the technique used in this thesis to estimate the mixing time of a Markov Chain and finally, we illustrate the computation of the mixing time of a Markov Chain with an example of cards shuffling. Finally, in Section 2.4, we show how to generate uniformly and at random a constrained vector by using both methods: recursive and MCMC (Markov Chain Monte Carlo), and we compare these two generation methods by mentioning their strengths and weaknesses.

2.2 Recursive Method

2.2.1 Principle

For a clear and detailed explanation of the recursive method, the reader is referred to [FS09]. The idea of the recursive method is to randomly generate using a combinatorial decomposition of the object. Once the decomposition is done, it is possible to build recursively the object. To understand more precisely the recursive method, we give an idea of this principle with the example of random generation of a binary tree.

2.2.2 Example of Binary Tree Recursive Generation

Binary Tree A binary tree is a rooted tree such that each node has 0, 1 or 2 children. The nodes without children are called leaves (external nodes) and the nodes that have children are called internal nodes. Let A be a binary tree with n internal nodes and v an internal node of A. We call the left subtree of v, denoted *Left*(v), the subtree of A having for root the left child of v (resp. right subtree). Therefore, the left subtree of A is the left subtree of its root, and the right subtree of A is the right subtree of its root. Let *size(Left)* be the size of the left subtree of

A (resp. *size(Right)*). Thus, the size of A is: $size(A) = 1 + size(Left) + size(Right)$.

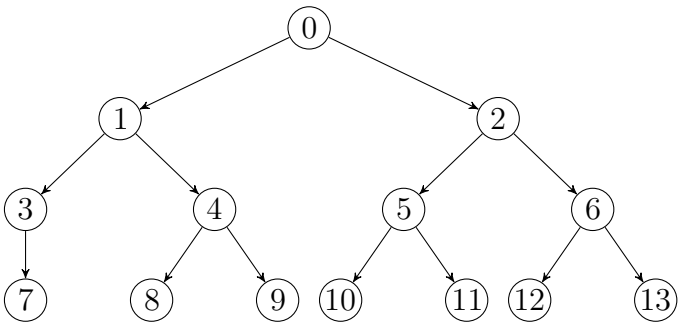


Figure 2.1: Example of binary tree with 14 nodes.

Example. *Figure 2.1 illustrates a binary tree with 14 nodes. Node 0 is the root node of the binary tree. Nodes 1 to 6 are the internal nodes. Nodes 7 to 13 are the leaves. The left subtree is the set of nodes* $\{1, 3, 4, 7, 8, 9\}$. *The right subtree is the set of nodes* $\{2, 5, 6, 10, 11, 12, 13\}$. *The size of the binary tree is* $1 + 6 + 7 = 14$.

Recursive Generation Suppose n is the size of a binary tree. The number of unlabeled binary trees of size n, B_n, can be recursively expressed by the following equation:

$$B_n = \sum_{i=1}^{n} B_{i-1} B_{n-i} \tag{2.1}$$

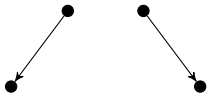
where $i - 1$ is the number of nodes in the left-subtree and $n - i$ is the number of nodes in the right-subtree. Note that $B_0 = 1$ because there is only one empty tree.

Example. *Let's calculate the number of unlabeled binary trees of size* $n = 4$ *using Equation 2.1.*

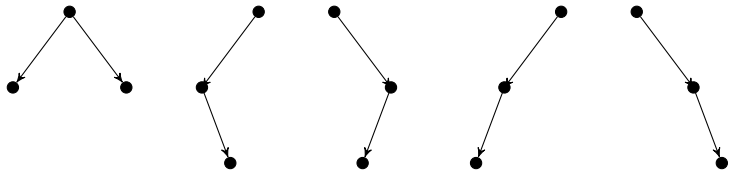
For $n = 0$, *there is only one empty tree, thus* $B_0 = 1$.

For $n = 1$, *we have one tree, thus* $B_1 = 1$.

For $n = 2$, $B_2 = B_0 B_1 + B_1 B_0 = 1 \times 1 + 1 \times 1 = 2$ *trees:*



For $n = 3$, $B_3 = B_0B_2 + B_1B_1 + B_2B_0 = 1 \times 2 + 1 \times 1 + 2 \times 1 = 5$ *trees:*



For $n = 4$, $B_4 = B_0B_3 + B_1B_2 + B_2B_1 + B_3B_0 = 1 \times 5 + 1 \times 2 + 2 \times 15 \times 1 = 14$ *trees.*

Note that we have counted the unlabelled binary trees. Thus, to count the labeled binary trees, each unlabelled binary tree can be labeled in $n!$ different ways.

Example. *Now let's calculate the number of labeled binary trees of size* $n = 4$ *using the direct formula:* $B_4 = 14 \times 24 = 336$ *labeled binary trees.*

The recursive method goes through two stages. In the first stage, we decompose the binary tree by computing B_i for all $i \in \{1, \ldots, n\}$. The calculation can be recursively done. Then, we apply Algorithm 2.1 (ComputeSizeLeft(n)) to compute the size k of the left-subtree. One can deduce that the size of the right-subtree is $n - 1 - k$. Algorithm 2.1 (ComputeSizeLeft(n)) starts by initializing to zero the

Algorithm 2.1: ComputeSizeLeft(n)

Input: B_i for all $i \in \{1, \ldots, n\}$
Output: The size k of left-subtree if tree not empty
$\perp$ otherwise.

```
1 begin
2     k = 0
3     s = B_{n-1} / B_n
4     r = random value ∈ [0, 1]
5     while s < r do
6         k = k + 1
7         s = s + B_k B_{n-k-1} / B_n
8     return k
```

size of the left-subtree. It samples uniformly a random value between zero and one,

and computes the ratio of the penultimate counted number of binary trees to the overall number of binary trees. Then, it checks if the ratio is less than the sampled value. While the condition is true, the algorithm iteratively increments the size of the left-subtree by one and updates at each iteration the computed ratio by adding to it the probability calculated in the Equation 2.2.

$$\mathbb{P}(\text{size(left)} = k) = \frac{B_k B_{n-k-1}}{B_n} \tag{2.2}$$

In the second stage, we apply Algorithm 2.2 (RecursiveBinaryTree(n)) to build the random object recursively. Algorithm 2.2 (RecursiveBinaryTree(n)) starts by recovering the size of the left-subtree (computed using Algorithm 2.1 ComputeSizeLeft(n)), then, it builds the left-subtree and right-subtree.

Algorithm 2.2: RecursiveBinaryTree(n)

Output: Binary tree if tree not empty
⊥ otherwise.

```
1 begin
2     if n = 0 then
3         return ⊥
4     k = ComputeSizeLeft(n)
5     Left = RecursiveBinaryTree(k)
6     Right = RecursiveBinaryTree(n − k − 1)
7     return the binary tree whose left subtree is Left and right subtree is
        Right
```

2.3 Markov Chain Method

In probability theory, Markov Chains constitute a practical probabilistic tool that applies to various fields. A Markov chain is a stochastic process describing a sequence of possible events in which the probability of each event depends only on the state attained in the previous event. Markov Chains play an important role for implementing random generators due to their relative ease of adaptation to different problems and their applicability to complex sampling problems.

Monte Carlo methods are about simulations. The concept is to use randomness to solve problems that might be deterministic in principle.

Markov Chain Monte Carlo (MCMC) methods are a combination of both techniques. They allow to simulate a distribution using Markov Chains. Hundreds of Markov Chain applications exist. MCMC are used in numerical approximation, for example in the estimation of kinetic parameters in systems biology [GR14]. Linguistics are interested in the relationship between languages and therefore in a form of phylogenesis, which also gives rise to the use of Markov Chains [NW08]. In computer science, we can find the application of MCMC in Machine Learning problems [EM12]. In bioinformatics, we can find MCMC in gene prediction. We can find the use of Markov Chains in internet applications like the PageRank of a webpage used by Google [Sul07].

2.3.1 Definitions

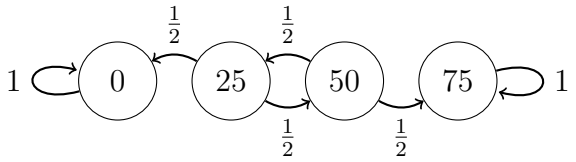
For a general reference on finite Markov Chains, the reader is referred to [LPW06]. A finite Markov Chain describes a process that randomly governs moving an element from a finite set Ω to another element of Ω. A rough vision of this process is to say that if x is an element of the set Ω then, starting from x, the next position will be determined by a probability $P(x, \cdot)$ defined on the set Ω. A square matrix M is stochastic if for each row i, $\sum_j M(i,j) = 1$ and if for all i, j, $0 \leq M(i,j) \leq 1$. Definition 2.1 formally specifies this property.

Definition 2.1 (Markov property)**.** *A* Markov Chain *on a finite space Ω with stochastic transition matrix M is a sequence $(X_i)_{i\in\mathbb{N}}$ of random variables on Ω satisfying for all $k > 1$, all $y, x_0, x_1 \ldots, x_k \in \Omega$:*

$$\begin{aligned}\mathbb{P}(X_k = y \mid X_0 = x_0, X_1 = x_1, \ldots, X_{k-1} = x_{k-1}) &= \mathbb{P}(X_k = y \mid X_{k-1} = x_{k-1}) \\ &= M(x_{k-1}, y).\end{aligned} \tag{2.3}$$

Equation 2.3 is called the Markov property. This property is translated by saying that a Markov Chain is memoryless. This expresses the fact that the distribution of the next position, given the current position and all past positions, does not depend on the already visited states, i.e. that the future only depends on the immediate past.

We can associate a directed graph G to the Markov Chain over the state space Ω. The set of vertices represents Ω, and for each $(x, y) \in \Omega^2$, (x, y) is an edge of G. The set of edges represents all the possible transitions between any pair of states. Each edge of the graph is weighted by a probability and the sum of the probabilities, for each vertex, on outgoing edges is equal to 1.



$$M = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0.5 & 0 & 0.5 & 0 \\ 0 & 0.5 & 0 & 0.5 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Figure 2.2: Graph and transition matrix for the gambler's ruin problem.

Example. *Figure 2.2 illustrates an introductory example of the gambler's ruin problem. Suppose a player has 25 euros. He tosses a coin until he loses or the coin comes up heads 2 times in a row. For each toss of coin, if the coin comes up heads he wins 25 euros and if it comes up tails he loses 25 euros.*

We can also note that for every natural number t, $M^t(x, y)$, the coefficient of the x^{th} row and y^{th} column of M^t is the probability (knowing that the current state is x) of changing from state x to state y in t steps [LPW06, Section 1.1]. If we have $M^t(x, y) > 0$, it means that a path of length t from x to y exists in the underlying graph associated with the related Markov Chain. In particular, there exists in G a cycle of length t passing through x, if and only if $M^t(x, x) > 0$.

Definition 2.2 (irreducibility)**.** *We consider M the transition matrix and G the graph underlying a Markov Chain on a state space Ω. We say that the chain M is irreducible if for all states x and y of Ω there exists a natural integer t such that $M^t(x, y) > 0$. The chain M is therefore irreducible if the graph G is strongly connected.*

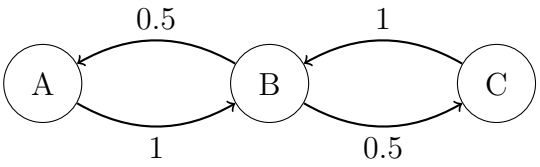
Example. *In Figure 2.2, M is not irreducible since starting from state 0 or 4, it is impossible to reach another state. But, in Figure 2.3, the Markov Chain is irreducible since each state can be accessed from any other state.*

Definition 2.3 (period of an element)**.** *We consider an element x of the state space Ω of a Markov Chain M. We call period of x, and we denote $\tau(x)$, the greatest common divisor of the set*

$$\left\{ t \in \mathbb{N} \setminus \{0\}, M^t(x, x) > 0 \right\}.$$

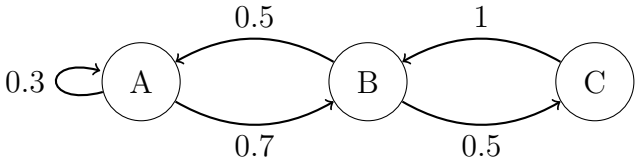
The period of x is also the greatest common divisor of the set of all the lengths of the cycles of G passing through x.

Example. *In Figure 2.3, the lengths of the cycles passing through A or C are 2 and 4. The greatest common divisor of 2 and 4 is 2. Therefore, the period of states A and C is 2. The lengths of the cycles passing through state B are 2. Thus, the period of state B is 2.*



$$M = \begin{pmatrix} 0 & 1 & 0 \\ 0.5 & 0 & 0.5 \\ 0 & 1 & 0 \end{pmatrix}$$

Figure 2.3: Irreducible periodic Markov Chain.



$$M = \begin{pmatrix} 0.3 & 0.7 & 0 \\ 0.5 & 0 & 0.5 \\ 0 & 1 & 0 \end{pmatrix}$$

Figure 2.4: Ergodic Markov Chain.

Note that if a Markov Chain M is irreducible, then, all states of M have the same period [LPW06, Lemma 1.6].

Definition 2.4 (period of irreducible chain)**.** *We consider M an irreducible Markov Chain. We call the period of M and we denote by $\tau(M)$ the period of all elements of the state space associated with M.*

Definition 2.5 (aperiodicity)**.** *We consider M an irreducible Markov Chain. We say that M is aperiodic if $\tau(M) = 1$.*

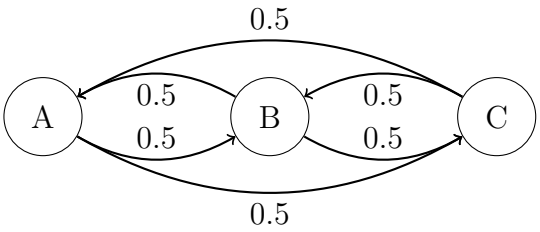
Example. *In Figure 2.4, the common period of all states is $\tau(M) = 1$, then the Markov Chain is aperiodic.*

Theorem 2.6 (ergodicity)**.** *A Markov Chain M is ergodic if it is irreducible and aperiodic.*

Example. *In Figure 2.4, the Markov Chain is irreducible since the underlying graph is strongly connected. The common period of all states is $\tau(M) = 1$, thus the chain is aperiodic. Therefore, the Markov Chain is ergodic.*

Definition 2.7 (symmetry)**.** *A symmetric matrix M is a square matrix that is equal to its transpose: $M = M^T$.*

We consider M the transition matrix and G the underlying graph of a Markov Chain on a state space Ω. The Markov Chain is symmetric if M is a symmetric matrix. In other words, if there is an edge (x, y) with probability p, then the graph has an edge (y, x) with the same probability.



$$M = \begin{pmatrix} 0 & 0.5 & 0.5 \\ 0.5 & 0 & 0.5 \\ 0.5 & 0.5 & 0 \end{pmatrix}$$

Figure 2.5: Symmetric Markov Chain.

Example. *In Figure 2.5, the transition matrix M is symmetric since $M = M^T$ and we can notice from the underlying graph that we can move between each connected pair of nodes with the same probability in both directions.*

Random mapping Finally, we will rely on a random mapping in Chapter 7 to build a specific Markov Chain. Consider a Markov Chain on a state space Ω with transition matrix M, and a set of independent and identically distributed random variables in a set Λ. Informally, a random mapping on Ω is defined by the function $\Omega \times \Lambda \to \Omega$. Intuitively, the function $\Omega \times \Lambda \to \Omega$ takes in the current state with some new random information, then it determines the next state of the Markov Chain.

2.3.2 Stationary Distribution

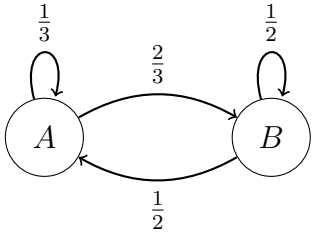
A stationary distribution of a Markov Chain is a probability distribution that remains unchanged in the chain as time progresses. The stationary distribution is represented as a row vector whose entries are probabilities summing to 1.

Definition 2.8 (stationary distribution)**.** *Let Ω be the state space and M the transition matrix associated with a Markov Chain. We consider π a probability law on Ω and we say that π is a stationary distribution (or invariant probability) if:*

$$\pi M = \pi. \tag{2.4}$$

In other words, if the chain starts at the date $t = 0$, with a distribution that corresponds to a stationary distribution ($\mu_0 = \pi$), then for every date t, we have $\mu_t = \pi$. Note that any finite-state Markov Chain has at least one stationary distribution [LPW06, Section 1.7].

Example. *To illustrate the stationary distribution, let us consider this example. Let M be the transition matrix for the Markov Chain on Figure 2.6 having two*



$$M = \begin{pmatrix} 1/3 & 2/3 \\ 1/2 & 1/2 \end{pmatrix}$$

Figure 2.6: Markov Chain with a stationary distribution.

states with a distribution π which assigns probability $\frac{3}{7}$ to the first state and $\frac{4}{7}$ to the second state. Let us check if π is stationary.

$$\pi M = \begin{pmatrix} \frac{3}{7} & \frac{4}{7} \end{pmatrix} \begin{pmatrix} \frac{1}{3} & \frac{2}{3} \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix} = \begin{pmatrix} \frac{3}{7} & \frac{4}{7} \end{pmatrix} = \pi.$$

So regardless the initial configuration of μ_0, the probability of finding a state in one of the 2 states tends to the stationary distribution as the number of steps in the MC grows.

Theorem 2.9 (unique stationary distribution, see [LPW06, Corollary 1.17])**.** *A Markov Chain has a unique stationary distribution if the Markov Chain is irreducible.*

Example. *In Figure 2.4, the Markov Chain is ergodic, therefore it has a unique stationary distribution. Contrarily, in Figure 2.2, the Markov Chain of the gambler's ruin problem has 2 stationary distributions concentrated at the states 0 and 75.*

Theorem 2.10 (uniform stationary distribution, see [LPW06, Section 1.6])**.** *If the Markov Chain is ergodic and symmetric, then the unique stationary distribution is uniform, i.e. walking enough steps in the graph leads to any state with the same probability whatever the starting vertex of the walk.*

Example. *An illustration example of the uniformity of the stationary distribution is depicted on Figure 2.7 where each reflexive transition has a probability $\frac{1}{2}$. Each other transition has a probability $\frac{1}{6}$. This Markov Chain is ergodic and symmetric thus it has a unique uniform stationary distribution. For instance, starting arbitrarily from the vertex A, after one step, we are in any other vertex with probability $\frac{1}{6}$ (and with probability 0 in the vertex A since there is no self-loop on it). After two steps, we are in the vertex A 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 vertex A is $p_{n+1} = \frac{1}{7} - \frac{1}{7}\left(\frac{-1}{6}\right)^n$ and is $\frac{1-p_{n+1}}{6}$ for all the other vertices. All probabilities tends to $\frac{1}{7}$ when n grows.*

Proof. Let A_n be the event: the chain is in A at step n ($p_n = p(A_n)$). If at step n we are in A, then $p_{n+1} = 0$. Otherwise, $p_{n+1} = \frac{1}{6}$.

$$p(A_{n+1}) = p(A_{n+1} \mid A_n)p(A_n) + p(A_{n+1} \mid \bar{A_n})p(\bar{A_n}).$$

We have: $p(A_{n+1} \mid A_n) = 0$, $p(A_{n+1} \mid \bar{A_n}) = \frac{1}{6}$, and $p(\bar{A_n}) = 1 - p_n$. Therefore:

$$p_{n+1} = \frac{1}{6}(1 - p_n).$$

It follows that:

$$\begin{aligned} p_{n+1} - \frac{1}{7} &= \frac{1}{6} - \frac{1}{6}p_n - \frac{1}{7} \\ &= -\frac{1}{6}\left(p_n - \frac{1}{7}\right) - \frac{1}{42} - \frac{1}{7} + \frac{1}{6} \\ &= -\frac{1}{6}\left(p_n - \frac{1}{7}\right). \end{aligned}$$

Consequently, $\left(p_{n+1} - \frac{1}{7}\right)$ is a geometric sequence and we have:

$$\begin{aligned} p_{n+1} - \frac{1}{7} &= \left(p_0 - \frac{1}{7}\right)\left(-\frac{1}{6}\right)^{n+1} \\ &= \left(1 - \frac{1}{7}\right)\left(-\frac{1}{6}\right)^{n+1} \\ &= \frac{6}{7}\left(-\frac{1}{6}\right)^{n+1} \\ &= \frac{1}{7}\frac{(-1)^{n+1}}{6^n}. \end{aligned}$$

It follows that $p_{n+1} = \frac{1}{7} - \frac{1}{7}\left(\frac{-1}{6}\right)^n$. □

Example. *An illustration example of the non-uniformity of the stationary distribution is depicted on Figure 2.2. The Markov Chain of the gambler is ruin is not irreducible since the state 25 cannot reach other states. Thus, the Markov Chain is not ergodic. Therefore, in this example we do not have a uniform stationary distribution.*

2.3.3 Total Variation Distance and Mixing Time

Total Variation Distance Let μ and ν be two probability distributions on a state space Ω. To know how far away from stationarity we are, we have to measure the total variation distance between μ and ν, assuming that one of these distributions is the stationary one and the other is the one at a given step.

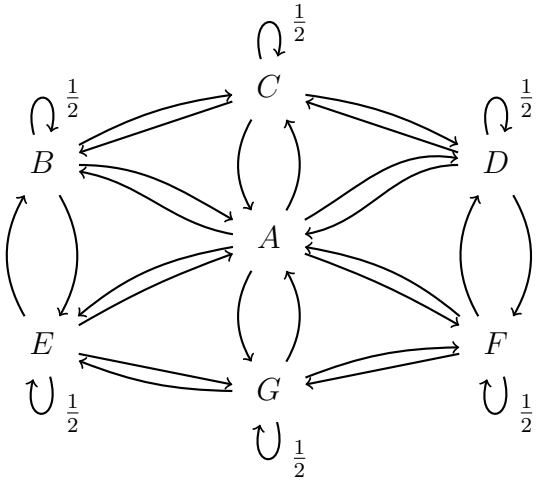


Figure 2.7: Example of the uniformity of the stationary distribution of a Markov Chain. The probability of each non-reflexive transition is $\frac{1}{6}$.

Definition 2.11 (total variation distance)**.** *The total variation distance between μ and ν is*

$$\|\mu - \nu\|_{TV} = \max_{A \subset \Omega} |\mu(A) - \nu(A)|. \tag{2.5}$$

Proposition 2.12. *Let μ and ν be two probability distributions on Ω. Then the total variation distance can be reduced to the sum:*

$$\|\mu - \nu\|_{TV} = \frac{1}{2} \sum_{x \in \Omega} |\mu(x) - \nu(x)|. \tag{2.6}$$

Example. *Let's assume that four tutorial sessions take place from 10- 12, 12-14, 14-16, and 16-18, respectively. The total number of attendees is 100. Let $\mu = (\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4})$ be the probability distribution of the hour slots. Let's assume that the number of attendees in each tutorial session hour slot is not distributed randomly at uniform. Let $\nu = (\frac{10}{100}, \frac{40}{100}, \frac{30}{100}, \frac{20}{100})$ be the probability distribution of the attendees. Then, the total variation distance between μ and ν is:*

$$\|\mu - \nu\|_{TV} = \frac{1}{2} \left(\left| \frac{1}{4} - \frac{10}{100} \right| + \left| \frac{1}{4} - \frac{40}{100} \right| + \left| \frac{1}{4} - \frac{30}{100} \right| + \left| \frac{1}{4} - \frac{20}{100} \right| \right) = 0.3.$$

Mixing Time When we perform a random walk on an ergodic Markov Chain, it is necessary to know when to stop moving, i.e. when to stop the process and return the current state. This issue is called mixing time, i.e. the number of steps required in order to be ε-close to the stationary distribution.

Definition 2.13 (mixing time)**.** *The mixing time* $t_{\text{mix}}(\varepsilon)$ *of an ergodic Markov Chain is defined by:*

$$t_{\text{mix}}(\varepsilon) = \min\{t \in \mathbb{N} : d(t) \leq \varepsilon\} \tag{2.7}$$

where $d(t)$ *is the distance from the stationary distribution as measured with the total variation distance.*

Bounding Mixing Time Practically, when using MCMC, it is very important to know how long does it take for an irreducible finite state Markov Chain to converge to its stationary distribution. If the chain does not run long enough, the distribution generated will differ significantly from the stationary distribution leading to invalid results. On the other hand, running the chain for too long can be very costly in terms of processing time. Thus, it is necessary to bound the mixing time of a Markov Chain. In general and unfortunately, finding a precise upper bound for the mixing time is very difficult to solve but progress has been made using analytic methods.

Note that, in this thesis, we are not interested in finding a lower bound on the mixing time since our objective is to detect the convergence of Markov Chains. Thus, we are only interested in finding an upper bound on the mixing time of Markov Chains.

Example. *The hypercube of dimension* n *is the graph whose vertices are the elements of the set* $\{0,1\}^n$*. Two vertices of the hypercube are connected by an edge if and only if they differ from only one coordinate. Figure 2.8 shows the hypercube of dimension* $n = 3$*. The random walk on* $\{0,1\}^n$ *moves from a vertex to one of its adjacent vertices by, uniformly at random, flipping a bit of the selected vertex.*

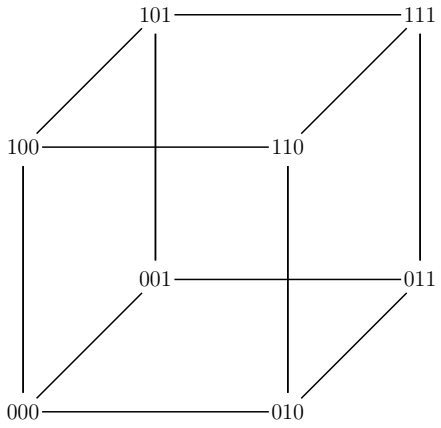


Figure 2.8: Hypercube of dimension $n = 3$.

For instance, if the random walk starts at the vertex having the coordinates 000, then we select uniformly at random one bit from 000. Suppose that the selected bit is the first 0 then it will be replaced by 1 and the move in the chain will be from vertex 000 to vertex 100. The process will be repeated until being close to the stationary distribution.

2.3.4 Statistical Tests

A frequently used approach to tackle the convergence of the Markov Chain consists in using statistical tests. It's about validating a hypothesis on mixing time in a generator that relies on a Markov Chain Monte Carlo algorithm. Statistical tests are tools that can be used to check if the quality of a sample generated with an MCMC algorithm is sufficient to provide an accurate approximation of the target distribution.

Among existing MCMC statistical tests, there is for instance Geweke diagnostic [GPH83], which divides the Markov Chain into 2 segments, then it computes the mean of each segment and finally detect the convergence according to the spectral density of the values representing the equality of means of both segments. Another statistical test is the autocorrelation test, which is the Pearson correlation [BCHC09] that indicates the extent to which two variables are linearly related. The test consists in recording the moves in the graph associated with a Markov Chain by associating them with a numerical value. This test calculates the autocorrelation of the state reached at the i^{th} step. If the values of the autocorrelation remain high when the number of steps increases, this indicates a long mixing time. The third example of MCMC statistical diagnostics is the Gelman-Rubin test [GR+92], which runs chains (at least two) of the same length $2n$, starting from distinct points. It rejects the first n steps of each chain and finally computes the potential scale reduction factor reflecting how much sharper the distributional estimate might become if the simulations were continued indefinitely.

In this thesis, we tried to use some statistical methods to detect the convergence of Markov Chains, but it did not lead to interpretable result. Due to the difficulty of using such methods, we finally decided to use a simpler visual method to detect the convergence of Markov Chains.

The approach is the following one: we start from different points of the state space (ideally well spread in the graph of the Markov Chain), then perform random walks from each starting point. At each iteration of the Markov process, we monitor a numerical parameter (a statistical measure). Finally, and after a long run, we detect the convergence of the Markov Chain by selecting the number of attempted

iterations when the curves of the different random walks overlap.

2.4 Random Constrained Vector Generation

To better understand the recursive and the MCMC methods, we generate (see [FS09, p. 39]) randomly and uniformly, using both methods, a vector $v \in \mathbb{N}^n$ satisfying:

$$\sum_{i=1}^{n} v(i) = N, \tag{2.8}$$

where N represents the sum of the vector v, n is the length of v and for vector $v = (v_1, \ldots, v_\ell)$, v_i is denoted $v(i)$. Moreover, we want to limit the maximum value in the vector v, which is useful to avoid large variance. For this purpose, we restrict the generation to a vector whose elements are in a controlled interval $[\alpha, \beta]$ such that $\alpha \leq \left\lfloor \frac{N}{n} \right\rfloor \leq \left\lceil \frac{N}{n} \right\rceil \leq \beta$ are positive integers.

2.4.1 Recursive Generation

As mentioned above, we want to generate uniformly and at random a constrained vector v using the recursive method as we did in [CESH18, Section III].

Let $h_{N,n}^{\alpha,\beta}$ be the cardinal of $H_{N,n}^{\alpha,\beta}$, the set of all vectors of size n, with sum N and minimum and maximum values, α and β, respectively. By decomposition, one has

$$h_{N,n}^{\alpha,\beta} = \sum_{k=\alpha}^{\beta} h_{N-k,n-1}^{\alpha,\beta}. \tag{2.9}$$

Moreover,

$$\begin{array}{l} h_{N,n}^{\alpha,\beta} = 0 \text{ if } \alpha n > N \text{ or } \beta n < N \text{ and,} \\ h_{N,1}^{\alpha,\beta} = 1 \text{ if } \alpha < N < \beta. \end{array} \tag{2.10}$$

In Equation 2.9, the first element of the vector is assumed to take each value between α and β. Algorithm 2.3 (Generate Sequences) uniformly generates a random vector over $H_{N,n}^{\alpha,\beta}$.

Example. *Assume that we want to generate uniformly at random, using Algorithm 2.3 (Generate Sequences), a vector v of size $n = 3$ and overall sum $N = 4$ where $v(i) \in \{1, 5\}$ for all $i \in \{1, \ldots, n\}$.*

Algorithm 2.3: Generate Sequences

Input: Integers N, n, α, β

Output: $v \in \mathbb{N}^n$ such that $\alpha \leq v(i) \leq \beta$ and $\sum_i v(i) = N$ if it is possible
$\perp$ otherwise.

begin
  if $\alpha > \beta$ ***or*** $n\alpha > N$ ***or*** $n\beta < N$ **then**
    return $\perp$
  for $1 \leq k \leq n$ ***and*** $0 \leq N' \leq N$ **do**
    compute $h^{\alpha,\beta}_{N',k}$ **using (2.9) and (2.10).**
  for $i \in [1, \ldots, n]$ **do**
    $s = 0$
    pick at random $v(i) \in [\alpha, \beta]$ **with** $\mathbb{P}(v(i) = k) = \dfrac{h^{\alpha,\beta}_{N-s-k,k}}{h^{\alpha,\beta}_{N-s,n-i}}$
    $s = s + v(i)$
  return v

The first step (precalculation) is the recursive decomposition of the vector v. By decomposition, we obtain the vector D such as:

$$D = [(1,0),(1,1),(1,2),(1,3),(1,4),(2,0),(2,1),(2,2),\\ (2,3),(2,4),(3,0),(3,1),(3,2),(3,3),(3,4)].$$

Each element of D has two values. The first value represents the size of the vector while the second one represents the sum. For instance, $D_9 = (2,3)$, the ninth element of D, represents the representations of 2 elements and of sum equals to 3. This corresponds to $H^{1,5}_{3,2}$.

The second step is enumaration. We count the number of representations of each element of D.

(1,1) : 1	(2,1) : 2	(3,1) : 3
(1,2) : 1	(2,2) : 3	(3,2) : 6
(1,3) : 1	(2,3) : 4	(3,3) : 10
(1,4) : 1	(2,4) : 5	(3,4) : 15

Table 2.1: The number of decomposition of each element of D into n elements such that $N_1 + N_2 + \ldots + N_n = N$. In this example, $\alpha = 1$ and $\beta = 5$.

For instance, $D_{10} = (2, 4)$ *and its representations are: (4,0), (0,4), (1,3), (3,1), and (2,2).*

The final step is the generation of the vector v*. According to Table 2.1,* $h_{4,3}^{1,5} = 15$*. These 15 objects can be decomposed into* γ *smaller classes, each one of size* γ_i*. We want to sample the integer* i *with a probability* $\frac{\gamma_i}{15}$ *where* $\sum \gamma_i = 15$*. We sample an integer in* $\{1, 15\}$ *and we compare it to* $\gamma_1, \gamma_1 + \gamma_2, \gamma_1 + \gamma_2 + \gamma_3$*, etc. as long as these values are smaller.*

For instance, we generate the following vector: $v = (1, 2, 1)$*.*

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, even with the rounded errors the random generation stays very close to the uniform distribution [DZ99].

Figure 2.9 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 2.10 shows the CV (Coefficient of Variation), which is the ratio of the standard deviation to the mean, 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 $\beta = 100$.

In the absence of constraints ($\alpha = 0$ and $\beta = N$), it is also possible to generate uniform vectors using Boltzmann samplers [DFLS04]: this approach consists in generating each $v(i)$ independently according to an exponential law of parameter γ. Theoretical results of [DFLS04] show that by choosing the right γ, the sum of the generated $v(i)$'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 2.9 in the unconstrained case. Moreover, in this case, Figure 2.10 shows that the CV is close to one, which is also the CV of an exponential distribution.

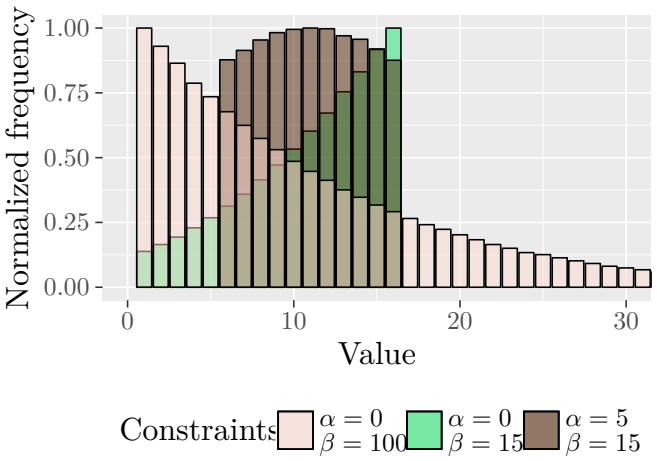


Figure 2.9: Frequency of each value in a vector of size $n = 10$ with $N = 100$ generated by Algorithm 2.3 (Generate Sequences) for three combinations of constraints for the minimum α and maximum β. For each case, the frequencies were determined by generating 100 000 vectors and are normalized to the maximum frequency. The frequency for large values when $\alpha = 0$ and $\beta = 100$ are not shown.

2.4.2 MCMC Generation

We want to generate uniformly at random a constrained vector using MCMC approach. Let n be the size of the vector, N the sum of its elements, and $\alpha \leq v(i) \leq \beta$ for all $1 \leq i \leq n$. v represents one vector of the state space of the Markov Chain. The first step is to arbitrary generate v such as $v(i) = \frac{N}{n}$. Then, we select uniformly at random 2 distinct elements from v. Let v_i and v_j be these elements. In order to move in the chain, we add 1 to v_i and subtract 1 from v_j iff the resulting elements v'_i and v'_j are respectively in $[\alpha, \beta]$. Thus, both selected elements v_i and v_j will be changed while the sum of the vector v remains unchanged (if the resulting elements v'_i and v'_j are not in $[\alpha, \beta]$, then we do not modify neither v_i nor v_j). We repeat the same process until the vector v is close to the stationary distribution.

The following is a formal proof of the uniformity of the random generation of a vector using MCMC approach.

Proof. The Markov Chain of the vector is aperiodic and its underlying graph is strongly connected, thus the Markov Chain is ergodic.

The aperiodicity of the Markov Chain is easy to prove. Practically, when we add or substract 1 from an element of our vector, it is possible to have rejections if the

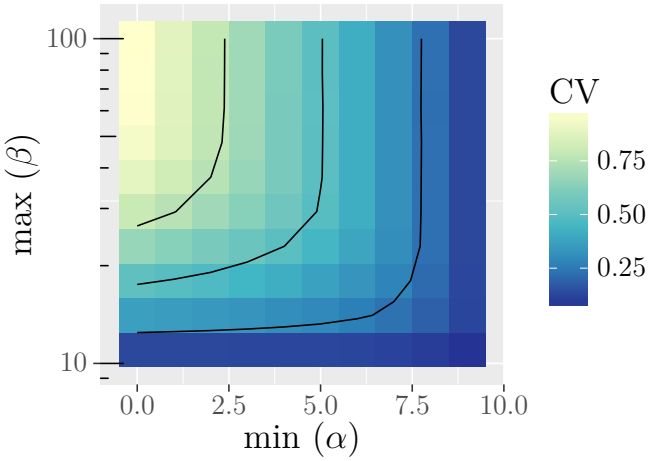


Figure 2.10: Mean CV in vectors of size $n = 10$ with $N = 100$ generated by Algorithm 2.3 (Generate Sequences) for different constraints for the minimum α and maximum β. Each tile corresponds to 10 000 vectors. The contour lines correspond to the levels in the legend (0.25, 0.5 and 0.75).

new value is not in $[\alpha, \beta]$. Thus, the Markov Chain is aperiodic.

The following is the proof of the strong connectivity of the underlying graph of the Markov Chain. This property is proved by induction (prove that a property holds for every natural number).

Let $v = v_1, \ldots, v_n$ and $v' = v'_1, \ldots, v'_n$. The distance between v and v' is:

$$d(v, v') = \sum_{i=1}^{n} |v_i - v'_i|.$$

One has $d(v, v') = 0$ iff $v = v'$. If $v \neq v'$, there exists i such that $v_i < v'_i$ (since $\sum v_i = \sum v'_i = N$) and j such that $v_j > v'_j$. Moreover, $\alpha \leq v_i < v'_i \leq \beta$ and $\alpha \leq v'_j < v_j \leq \beta$.

Let $v'' = (v_1, \ldots, v_{i-1}, v_i + 1, v_{i+1}, \ldots, v_j, v_j - 1, v_{j+1}, \ldots)$, where all elements of v'' are in $[\alpha, \beta]$.

There is an edge from v to v'' in the underlying graph of the Markov Chain. Moreover, $d(v'', v') < d(v, v')$. Thus, the construction of a path from v to v' can be done by an easy induction on $d(v, v')$. Therefore, the underlying graph of the Markov Chain is strongly connected and the Markov Chain is ergodic.

In addition, the transition matrix of the Markov Chain is symmetric since for all states the probability of the transition from a state to another one in both

directions is the same. More specifically, the probability of any transition is $\frac{1}{n(n-1)}$, i.e. the probability of choosing a given i and j in Algorithm 2.4 (Constrained Vector MCMC Generation).

Therefore, the distribution of vectors generated with constraints using MCMC approach is uniform. □

Algorithm 2.4 (Constrained Vector MCMC Generation) generates a random vector using MCMC approach.

Algorithm 2.4: Constrained Vector MCMC Generation

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

```
1  begin
2      if α > β or nα > N or nβ < N then
3          return ⊥
4      for i ∈ [1, ..., n] do
5          v(i) ← N/n
6      iter ← 1
7      while iter ≤ nbIterations do
8          i ← rand(1, n)
9          j ← (rand(0, n − 2) + i)  mod n + 1
10         if (v(i) + 1) ≤ β and (v(j) − 1) ≥ α then
11             v(i) ← v(i) + 1
12             v(j) ← v(j) − 1
13         iter ← iter + 1
14     return v
```

Example. *Assume that we want to generate, using Algorithm 2.4 (Constrained Vector MCMC Generation), a vector v of size $n = 3$, an overall sum $N = 9$, where $v(i) \in \{1, 5\}$ for all $i \in \{1, \ldots, n\}$, and the number of iterations is 100.*

The initial vector is $v = (3, 3, 3)$. As we see in Algorithm 2.4 (Constrained Vector MCMC Generation), at each iteration we sample uniformly at random 2 elements of v and modify their values by adding one to an element and substracting one from the second element. The choices to modify 2 elements of v are the following ones: [(-1,0,1), (-1,1,0), (0,1,-1), (0,-1,1), (1,-1,0), (1,0,-1)]. Assume that we

select randomly at uniform the choice (-1,1,0). We add the sampled choice to v (if all elements of v remain in $\{1,5\}$), thus the new vector v, after the first iteration will be: $v = (2,4,3)$. We repeat the process until reaching the number of iterations. Finally and after 100 iterations, we will obtain for instance the vector $v = (2,6,1)$.

Practically, the Markov Chain can be modified to mix faster: rather than changing each element by $+1$ or -1, each element of v can be modified by $+k$ or $-k$ such that the constraints given by α and β are valid.

Algorithm 2.5 (Constrained Vector Faster MCMC Generation) generates a random vector using faster MCMC approach.

Algorithm 2.5: Constrained Vector Faster MCMC Generation

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

begin
  if $\alpha > \beta$ **or** $n\alpha > N$ **or** $n\beta < N$ **then**
    return $\perp$
  for $i \in [1, \ldots, n]$ **do**
    $v(i) \leftarrow \frac{N}{n}$
  $iter \leftarrow 1$
  while $iter \leq nbIterations$ **do**
    $i \leftarrow rand(1, n)$
    $j \leftarrow (rand(0, n-2) + i) \mod n + 1$
    $k \leftarrow rand(\max(\alpha - v(i), v(j) - \beta), \min(v(j) - \alpha, \beta - v(i)))$
    if $(v(i) + k) \leq \beta$ **and** $(v(j) - k) \geq \alpha$ **then**
      $v(i) \leftarrow v(i) + k$
      $v(j) \leftarrow v(j) - k$
    $iter \leftarrow iter + 1$
  return v

2.4.3 Recursive vs MCMC

Recursive vs mcmc vs mcmcFaster In order to compare both methods of generation, and to show when both methods converge to a uniform distribution

(distribution obtained by the recursive method), we illustrate on Figure 2.11 the comparison with the example of 1000 vectors of size $n = 100$ with $N = 1000$, $\alpha = 5$ and $\beta = 30$. Vectors generated using mcmc and mcmcFaster approaches are respectively generated with Algorithms 2.4 (Constrained Vector MCMC Generation) and 2.5 (Constrained Vector Faster MCMC Generation) while vectors generated using the recursive method are generated with Algorithm 2.3 (Generate Sequences). Note that we assume that studying the variance of the costs provides a good

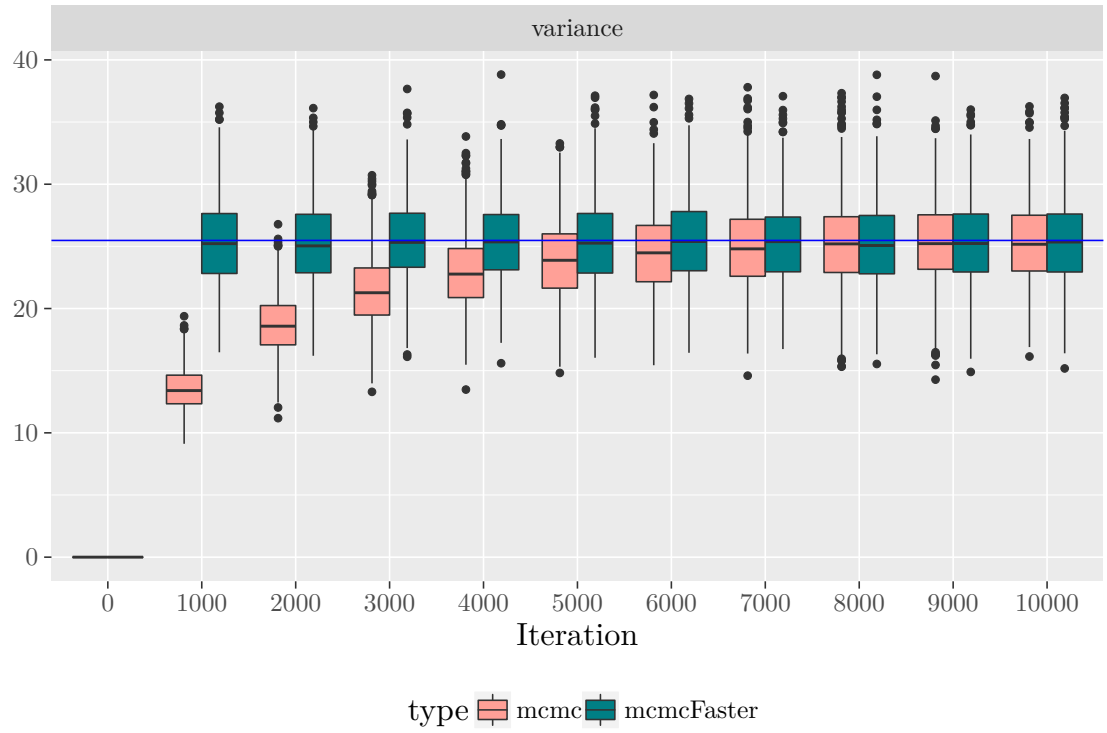


Figure 2.11: Evolution of the variance for vectors of size $n = 100$, with sum of elements $N = 1000$, minimum value of elements $\alpha = 5$, maximum value of elements $\beta = 30$. Vectors generated using mcmc and mcmcFaster approaches are generated with Algorithms 2.4 (Constrained Vector MCMC Generation) and 2.5 (Constrained Vector Faster MCMC Generation). Each boxplot corresponds to 1000 vectors. The horizontal blue line corresponds to the mean of variances of 1000 vectors generated with Algorithm 2.3 (Generate Sequences) with the same characteritics of vectors in boxplots.

indication of whether the stationary distribution is reached.

First, we recursively generate 1000 vectors, compute the variance of each vector, then compute the mean of variances, which is equal to 26 (represented by the blue

line). Second, for mcmc and mcmcfaster methods, we arbitrary generate an initial vector where each of its elements is equal to $\frac{N}{n}$ and we monitor the variance of each vector. Next, 10 000 walks are performed and the value of the variance is reported after every 1 000 iterations.

We can observe that the pair of boxplots[1] are synchronized with the uniform distribution after about 8 000 iterations. In other words, in this example, we illustrate the mcmc and mcmcfaster seems to be close to the uniform distribution after about 8 000 iterations. Boxplots corresponding to mcmcFaster method tend to the uniform distribution after about 1 000 iterations while boxplots corresponding to mcmc method tend to the uniform distribution after about 8 000 iterations. This result illustrates that using mcmcFaster method to generate constrained vectors leads to rapidly mixing the Markov Chain.

Advantages and limitations Markov Chain Monte Carlo and the recursive random generation are powerful methods. Both approaches have gained widespread use due to their advantages but at the same time each method has some limitations. In Table 2.2, we list the most well-known advantages and limitations of the MCMC and recursive random generation methods.

	MCMC	**Recursive**
Advantages	Relatively simple to implement	Generally computationally efficient
	Applicable to complex sampling problems	Avoids convergence issues
Limitations	Can be very difficult to assess accuracy and evaluate convergence, even empirically	Can be very difficult to find the calculation formulas
	Has a burn-in phase, which starts with arbitrary position and ends up with reaching the target distribution. This burn-in phase usually takes a long time, especially when the distribution is defined over high dimensional space	The precalculation can take a long time if the size of the object is too big

Table 2.2: Advantages and limitations of MCMC and recursive generation.

[1]Each boxplot consists of a bold line for the median, a box for the quartiles, whiskers that extend at most to 1.5 times the interquartile range from the box and additional points for outliers.

Part II

Directed acyclic graphs

Chapter 3

Motivation and Problem Statement for DAGs

Contents

3.1 Introduction

In the context of parallel systems, instances for numerous multiprocessor scheduling problems contain the description of an application to be executed on a platform [Leu04]. This part focuses on scheduling problems requiring a Directed Acyclic Graph (DAG) as part of the input. Such a DAG represents a set of tasks to be executed in a specific order given by precedence constraints: the execution of any task cannot start before all its predecessors have completed their executions. Scheduling

a DAG on a platform composed of multiple processors consists in assigning each task to a processor and in determining a start time for each task. While this work studies the DAG structure for several scheduling problems, it illustrates and analyzes existing generation methods in light of a specific problem with unitary costs and no communication. This simple yet difficult problem emphasizes the effect of the DAG structure on the performance of scheduling heuristics.

Section 3.2 is dedicated to listing DAG properties. In Section 3.3, we describe the tools that have been proposed in the literature to generate DAGs in the context of scheduling in parallel systems. Then, we mention the existing sets of task graphs and finally we describe the Layer-by-Layer methods and the uniform random method to generate DAGs. Section 3.4 explains the different meanings of uniformity in the context of DAG generation. Finally, in Section 3.5, we describe the studied scheduling problem for DAGs in this thesis.

3.2 Properties and Notations

All graphs considered throughout this thesis are finite. A *directed graph* is a pair (V, E) where V is a finite set of *vertices* and $E \subseteq V \times V$ is the set of *edges*. A *path* is a finite sequence of consecutive edges, that is a sequence of the form $(v_1, v_2), (v_2, v_3), \ldots, (v_{k-1}, v_k)$; k is the *length* of the path, i.e. the number of vertices on this path.

The *output degree* of a vertex v is the cardinal of the set $\{(v, w) \mid w \in V,\ (v, w) \in E\}$. Similarly the *input degree* of a vertex v is the cardinal of the set $\{(w, v) \mid w \in V,\ (w, v) \in E\}$. The *output* (resp. *input*) *degree* of a directed graph is the maximum value of the output (resp. input) degrees of its vertices. The degree of a vertex is the sum of its input and output degrees.

A directed graph is *acyclic* (DAG for short) if there is no path of strictly positive length k such that $v_1 = v_k$ (with the above notation). Let $\mathcal{D}^n$ be the set of all DAGs whose set of vertices is $\{1, 2, \ldots, n\}$. In a DAG, if (v, w) is an edge, v is a *predecessor* of w and w a *successor* of v.

In a DAG D with n vertices, all paths have a length less than or equal to n. The *length* of a DAG is defined as the maximum length of a path in this DAG. The *depth* of a vertex v in a DAG is inductively defined by: if v has no predecessor, then its depth is 1; otherwise, the depth of v is one plus the maximum depth of its predecessors.

Note that the following properties that we introduce are useful to characterize

DAGs for the scheduling problem defined in Section 3.5.

The *shape decomposition* of a DAG is the tuple $(X_1, X_2, \ldots, X_k)$ where X_i is the set of vertices of depth i. Note that k is the length of the DAG. The *shape* of the DAG is the tuple $(|X_1|, \ldots, |X_k|)$. The maximum (resp. minimum) value of the $|X_i|$ is called the *maximum shape* (resp. *minimum shape*) of the DAG. Computing the shape decomposition and the shape of a DAG is easy. If $|X_i| = 1$, the unique vertex of X_i is called a *bottleneck vertex*.

A *block* is a subset of vertices of the form $\cup_{i<j<i+\ell} X_j$ with $\ell > 1$ where X_i is either a singleton or $i = 0$, $X_{i+\ell}$ is either a singleton or $i + \ell = k + 1$, and for each $i < j < i + \ell$, $|X_j| \neq 1$.

We denote by $\text{mass}^{\text{abs}}(B)$ the cardinal of $B = \cup_{i<j<i+\ell} X_j$ and by $\text{mass}^{\text{abs}}(D) = \max\{\text{mass}^{\text{abs}}(B) \mid B \text{ is a block}\}$ the *absolute mass* of D. The *relative mass*, or simply the *mass*, is given by $\text{mass}(D) = \frac{\text{mass}^{\text{abs}}(D)}{n}$.

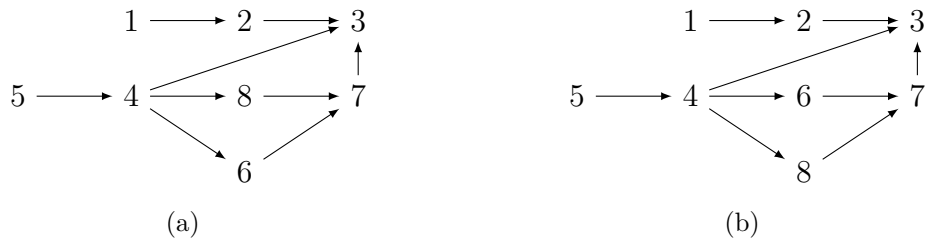


Figure 3.1: Examples of DAGs.

Example. *The following properties belong to the DAG on Figure 3.1(a):*

- *The predecessors of vertex 3 are vertices 2, 4, and 7.*
- *The successor of vertex 2 is vertex 3.*
- *A longest path is* $(5, 4), (4, 6), (6, 7), (7, 3)$*, its length is 5.*
- *The output degree of vertex 4 is 3.*
- *The input degree of vertex 7 is 2.*
- *The depth of the vertex 8 is 3.*
- *The shape decomposition is the tuple* $(\{1, 5\}, \{2, 4\}, \{6, 8\}, \{7\}, \{3\})$.
- *The shape is the tuple* $(2, 2, 2, 1, 1)$.
- *It has two bottleneck vertices* 7 *and* 3.
- *It contains only one block of length* $l = 4$

- *Its absolute mass is* $2 + 2 + 2 = 6$.

In a DAG, two distinct vertices v and w are *incomparable* if there is neither a path from v to w, nor from w to v.

The *width* of a graph is the maximum size of the subset of vertices whose elements are pair-wise incomparable. Since vertices of same depth are incomparable, the maximum shape of a DAG is less than or equal to its width. In the context of scheduling, the width represents the maximum number of vertices (tasks) that can be executed in parallel. It is also the size of the largest antichain[1], which can be computed in polynomial time using Dilworth's theorem and a technique developed by Ford and Fulkerson [FJF16]. The methodology is conjectured to have a time complexity of $O(n^{5/2})$ [Plo07]. In some cases (for instance the *comb* DAG, see Section 4.1), the width can be much larger than the maximum shape.

Table 3.1 compares the width and the maximum shape on the DAGs obtained with two random generators explored in this part. The first generator is the Erdős-Rényi where an upper-triangular adjacency matrix is randomly generated. For each pair of vertices (i, j) with $i < j$, there is an edge from i to j with an independent probability p (see Section 4.2.1). The second generator generates DAGs uniformly at random using recursive approach (see Section 4.2.2).

Example. *Consider the DAG on Figure 3.1(a). The vertices that can be executed in parallel are: (1,5), (2,4), and (6,8). Therefore, the width of the DAG is 2.*

n	Erdős-Rényi	Uniform
10	2.95 – 0.34 – 2	2.35 – 0.09 – 1
20	3.52 – 0.45 – 2	2.77 – 0.14 – 1
30	3.62 – 0.46 – 2	3.13 – 0.23 – 1

Table 3.1: Comparison of width and maximum shape of randomly generated DAGs with different methods: "Erdős-Rényi" for the so-called algorithm with parameter $p = 0.5$ (see Section 4.2.1) and "Uniform" for the recursive random generator (see Section 4.2.2). Reported numbers $x - y - z$ correspond respectively to the average width, the average difference between width and shape width, and the maximum difference pointed out. Each experiment is performed by sampling 100 DAGs.

Two DAGs (V_1, E_1) and (V_2, E_2) are isomorphic, denoted $(V_1, E_1) \sim (V_2, E_2)$, if there exists a bijective map φ from V_1 to V_2 such that $(x, y) \in E_1$ iff $(\varphi(x), \varphi(y)) \in$

[1]Set of incomparable vertices (for which no path exists between any pair of vertices).

E_2. The relation $\sim$ is an equivalence relation. Intuitively, two DAGs are isomorphic if they are equal up to vertices names.

Example. *The DAGs on Figure 3.1 are isomorphic.*

The *transitive reduction* of a DAG D [AGU72] is the DAG D^T for which: D^T has a directed path between u and v iff D has a directed path between u and v; there is no graph with fewer edges than D^T that satisfies the previous property. Intuitively, this operation consists in removing redundant edges.

Example. *The transitive reduction of the DAG on Figure 3.1(a) is:*

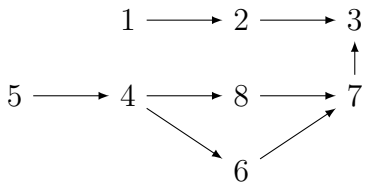


Figure 3.2: Transitive reduction of the DAG on Figure 3.1(a)

The *reversal* of a DAG D is the DAG D^R for which there is an edge between u and v iff there is an edge between v and u in D. Intuitively, this operation consists in reversing the DAG.

Example. *The reversal DAG of the DAG on Figure 3.1(a) is:*

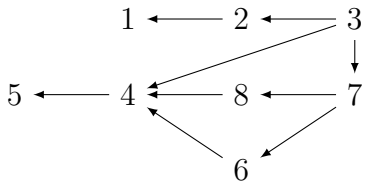


Figure 3.3: Reversal DAG of the DAG on Figure 3.1(a)

Finally, Table 3.2 presents some of the DAG properties that may impact the performance of scheduling algorithms. We discard the minimum input and output degrees because they are always $\deg_{\text{in}}^{\text{min}} = \deg_{\text{out}}^{\text{min}} = 0$. We also discard the mean input and output degrees because they are always equal to half the mean degree ($\deg_{\text{in}}^{\text{mean}} = \deg_{\text{out}}^{\text{mean}} = \frac{\deg^{\text{mean}}}{2}$). For all nine edge-related properties (m and the degree-based properties) applied to a DAG D, we can also compute them on the

transitive reduction D^T. The vertex-related properties (n, the width and the shape-based ones) remain the same on the transitive reduction. For all seven shape-based properties on a DAG D, we can also compute them on the reversal D^R. The edge-related properties remain the same through the reversal with the inversion of $\deg_{\text{in}}^{\max}$ and $\deg_{\text{in}}^{\text{sd}}$ with $\deg_{\text{out}}^{\max}$ and $\deg_{\text{out}}^{\text{sd}}$, respectively. Finally, some of these properties are related: $n \times \deg^{\text{mean}} = \frac{m}{2}$ and $\text{len} \times \text{sh}^{\text{mean}} = n$.

Symbol	Definition
n	number of vertices
m	number of edges
$\deg^{\max}(\deg_{\text{in}}^{\max}, \deg_{\text{out}}^{\max})$	maximum (input, output) degree
$\deg^{\min}$	minimum degree
$\deg^{\text{mean}}$	mean (input, output) degree
$\deg^{\text{sd}}(\deg_{\text{in}}^{\text{sd}}, \deg_{\text{out}}^{\text{sd}})$	standard deviation of the (input, output) degrees
len or k	length (also called height, number of levels, longest path or critical path length)
width	width
$\text{sh}^{\max}$	maximum shape
$\text{sh}^{\min}$	minimum shape
sh^{mean}	mean shape (parallelism in [TK02])
sh^{sd}	standard deviation of the shape
sh^{1}	number of source vertices (vertices with null input degree)
sh^{k}	last element of the shape
mass	(relative) mass
p	connectivity probability
K	number of permutations (for the random orders method in Section 4.2.3)
P	set of processors

Table 3.2: List of DAG properties and other notations. When necessary, we specify on which DAG a property is measured (e.g. $m(D^T)$ for the number of edges in the transitive reduction of D).

3.3 Generation of DAGs

Our approach is similar to the one followed in [CMP+10] and [Mar18], which consists in studying the properties of randomly generated DAGs before comparing

the performance of scheduling heuristics. In [CMP+10], three properties are measured and analyzed for each studied generation method: the length of the longest path, the distribution of the output degrees and the number of edges. We describe 15 such properties in Table 3.2. The authors consider five random generation methods (described in this section and Section 4.2): two variants of the Erdős-Rényi algorithm [ER59], one layer-by-layer variant [ACD74], the random orders method [Win85] and the Fan-in/Fan-out method [CMP+10]. Finally, for each generation method, we compare the performance of four scheduling heuristics. The results of [CMP+10] are consistent with the observations done in Section 4.2 (Figures 4.2, 4.5 and 4.8) for the length and the number of edges.

A similar approach is undertaken in [Mar18]. First, three characteristics are considered: the number of vertices in the critical path, the width (or maximum parallelism) and the density of the DAG in terms of edges. These characteristics are studied on DAGs generated by two main approaches (the Erdős-Rényi algorithm and a MCMC approach) with sizes between 5 and 30 vertices. Finally, although no DAG property is studied, scheduling heuristics are compared using a variety of random and non-random DAGs in [KA99].

We describe below generation tools, data sets and random generation methods.

3.3.1 Tools for DAG Generation

Many tools have been proposed in the literature to generate DAGs in the context of scheduling in parallel systems. TGFF (Task Graphs For Free)[2] is the first tool proposed for this purpose [DRW98]. This tool relies on a number of parameters related to the task graph structure: maximum input and output degrees of vertices, average for the minimum number of vertices, etc. The task graph is constructed by creating a single-vertex graph and then incrementally augmenting it. Until the number of vertices in the graph is greater than or equal to the minimum number of vertices, this approach randomly alternates between two phases: the expansion of the graph and its contraction. The main goal of TGFF is to gain more control over the input and output degrees of the tasks.

DAGGEN[3] was later proposed to compare heuristics for a specific problem [DNSC09]. This tool relies on a layer-by-layer approach with five parameters: the number of vertices, a width and regularity parameters for the layer sizes, and a density and jump parameters for the connectivity of the DAG. The number of

[2] http://ziyang.eecs.umich.edu/projects/tgff/index.html
[3] https://github.com/frs69wq/daggen

elements per layer is uniformly drawn in an interval centered around an average value determined by the width parameter and with a range determined by the regularity parameter. Lastly, edges are added between layers separated by a maximum number of layers determined by the jump parameter (edges only connect consecutive layers when this parameter is one). For each vertex, the method adds a uniform number of predecessors in an interval determined by the density parameter.

GGen[4] has been proposed to unify the generation of DAGs by integrating existing methods [CMP+10]. The tool implements two variants of the Erdős-Rényi algorithm, one layer-by-layer variant, the random orders method and the Fan-in/Fan-out method. It also generates DAGs derived from classical parallel algorithms such as the recursive Fibonacci function, the Strassen multiplication algorithm, the Cholesky factorization, etc.

The Pegasus workflow generator[5] can be used to generate DAGs from several scientific applications [JCD+13] such as Montage, CyberShake, Broadband, etc. XL-STaGe[6] produces layer-by-layer DAGs using a truncated normal distribution to distribute the vertices to the layers [CDB+16]. This tool inserts edges with a probability that decreases as the number of layers between two vertices increases. A tool named RandomWorkflowGenerator[7] implements a layer-by-layer variant [GCJ17]. Other tools have also been proposed but are no longer available as of this writing: DAGEN [AM11], RTRG[8] [SAHR12], MRTG [AAP+16].

Finally, other fields such as electronic circuit design or dataflow also use DAGs. In this last field, however, requirements differ: the acyclicity is no longer relevant, while ensuring a strong connectedness is important. Two noteworthy generators have been proposed: SDF^3 inspired from TGFF[9] [SGB06] and Turbine[10] [BLDMK14].

3.3.2 Instance Sets

The STG (Standard Task Graph) set[11] has been specifically proposed for parallel systems [TK02] and is frequently used to compare scheduling heuristics [AKN05, DŠTR12]. The DAG structures of STG relies on four different methods.

[4] https://github.com/perarnau/ggen
[5] https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator
[6] https://github.com/nizarsd/xl-stage
[7] https://github.com/anubhavcho/RandomWorkflowGenerator
[8] http://users.ecs.soton.ac.uk/ras1n09/rtrg/index.html (unavailable as of this writing)
[9] http://www.es.ele.tue.nl/sdf3/
[10] https://github.com/bbodin/turbine
[11] http://www.kasahara.elec.waseda.ac.jp/schedule/

Two methods, `sameprob` and `samepred`, rely on the Erdős-Rényi algorithm, while the other two, `layrprob` and `layrpred`, constitute layer-by-layer variants. A connection probability is given to `sameprob` and `layrprob`, while an average number of predecessors is given to `samepred` and `layrpred`. With these last two methods, the parameter is apparently converted to a connection probability inferred from the size of the DAG. Any layer-by-layer variant proceeds by first distributing vertices into layers such that the average layer size is 10. Then, edges between any pair of vertices from distinct layers are added from top to bottom according to the connectivity parameter. The size of the DAGs varies from 50 to 5,000. For each size, the data set contains 15 instances for each combination of a method among the four ones and a value for the connectivity parameter among three possible ones (leading to 180 instances). Both layer-by-layer variants do not guarantee that the layer of any vertex equals its depth. As a consequence, the length is not necessarily $\frac{n}{10} + 2$ (2 dummy vertices are always added) where n is the number of vertices[12] and this problem becomes more apparent with large DAGs generated by `layrpred` because there are not enough inserted edges to ensure the layered structure. The STG set also contains costs and real DAGs such as robot control, sparse matrix solver and SPEC fpppp program.

PSPLIB[13] contains difficult instances for RCPSP (Resource-Constrained Project Scheduling Problems) [KSD95], a scheduling problem in the field of project management. Finally, in the graph drawing context, a set of 112 real-life graphs were proposed[14] [DBGL+97] but are no longer available.

In addition to those implemented in GGen and the ones in STG, other DAGs from real-cases can be used such as the LU decomposition [LKK83], the parallel Gaussian elimination algorithm [CMRT88], the parallel Laplace equation algorithm [WG90], the mean value analysis (MVA) [AVÁM92], which has a diamond-like structure, the FFT algorithm [CLRS09], which has a butterfly structure, the QR factorization, etc.

3.3.3 Layer-by-Layer Methods

The layer-by-layer method was first proposed by [ACD74] but popularized later by the introduction of the STG data set [TK02]. This method produces DAGs in which vertices are distributed in layers and vertices belonging to the same layer are independent. The method consists in three steps: determining the number of

[12] This is the case for the instance `rand0038.stg` for size 50.
[13] `http://www.om-db.wi.tum.de/psplib/`
[14] `ftp://infokit.dis.uniromal.it/public/` (unavailable as of this writing)

layers; distributing the vertices to the layers; connecting the vertices from different layers. In most proposed methods, there is at least one parameter for each step. For instance, the shape parameter controls the number of layers and is related to the ratio of $\sqrt{n}$ to the number of layers [THW02a, IT07, GCJ17].

The number of layers can be drawn from a parameterized uniform distribution [ACD74, THW02a, IT07, SMD11], given as a parameter [CMP+10, GCJ17] or generated in a non-parameterized way [AK98, TK02, CDB+16].

Similarly, vertices can be distributed by generating a number of vertices at each layer with a parameterized uniform distribution [ACD74, THW02a, IT07, DNSC09, SMD11], by selecting a layer for each vertex with a parameterized normal distribution [CDB+16], by using a balls into bins approach [CMP+10, GCJ17] or in a non-parameterized way [AK98]. Note that generating a uniform number of vertices per layer may lead to a different number of vertices n than expected. Also, using a balls into bins strategy may lead to empty layers.

Finally, the connection between vertices can depend on a connection probability [TK02, DNSC09, CMP+10, SMD11, CDB+16] or an average number of predecessors or successors for each vertex [ACD74, TK02, THW02a]. Although vertices in the same layer may have different depth (e.g. this occurs in the STG data set), adding specific edges prevents this situation [DNSC09, GCJ17]. The layer-by-layer approach can also lead to DAGs with multiple connected components except for [ACD74]. Finally, some methods allow edges between non-consecutive layers [ACD74, TK02, CMP+10], while others limit them [DNSC09, CDB+16, GCJ17].

3.3.4 Uniform Random Generation

Many works address the problem of randomly generating DAGs. Uniform random generation of DAGs can be done using counting approaches [Rob73] based on generating functions. Many existing methods have been developped in the literature and the most important ones are described in Section 4.2.

While previous uniform approaches consider only the size of the DAG n as a parameter, other studies have proposed to generate directed graphs from a prescribed degree sequence [MKI+03, KN09, AAK+13]. A uniform method is proposed in [MKI+03] but may produce cyclic graphs. In contrast, the method proposed in [KN09] forbids cyclicity but has no uniformity guarantee. Last, in the context of sensor streams, several methods has been proposed [AAK+13] to generate DAGs with a prescribed degree distribution.

Finally, a multitude of related approaches has been proposed but are discarded in this study because of their specificity. For instance, specific structures may be used to assess the performance of scheduling methods [LALG13, CMSV18] or special DAGs with known optimal solutions relatively to a given platform may also be built [KA99, OVRO+18].

3.4 Uniformity of the Random Generation

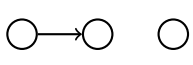
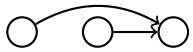
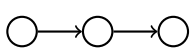
Isom. classes	Matrices	ER	Labeling
	$\begin{pmatrix} 0 & 0 \\ & 0 \end{pmatrix}$	$\frac{1}{8}$	1
	$\begin{pmatrix} 1 & 0 \\ & 0 \end{pmatrix}$ $\begin{pmatrix} 0 & 1 \\ & 0 \end{pmatrix}$ $\begin{pmatrix} 0 & 0 \\ & 1 \end{pmatrix}$	$\frac{3}{8}$	6
	$\begin{pmatrix} 0 & 1 \\ & 1 \end{pmatrix}$	$\frac{1}{8}$	3
	$\begin{pmatrix} 1 & 1 \\ & 0 \end{pmatrix}$	$\frac{1}{8}$	3
	$\begin{pmatrix} 1 & 0 \\ & 1 \end{pmatrix}$	$\frac{1}{8}$	6
	$\begin{pmatrix} 1 & 1 \\ & 1 \end{pmatrix}$	$\frac{1}{8}$	6

Table 3.3: DAGs with 3 vertices: there is one row for each isomorphism class. For each class, we report: all corresponding (upper triangular) adjacency matrices; the probability of generating such a DAG with the Erdős-Rényi algorithm ($p = 0.5$); and, the number of DAGs in each isomorphism class (i.e. the number of labelings).

This work focuses on the importance of generating DAGs uniformly. We discuss the notion of uniformity through the example with 3 vertices given in Table 3.3. In this instance, there are six isomorphism classes (i.e. six different unlabeled DAGs) for a total of 25 different (labeled) DAGs. A generator is thus uniform up to isomorphism if it generates each isomorphism class (or unlabelled DAGs) with a probability $\frac{1}{6}$ or uniform on all (labelled) DAGs if it generates each DAG with a probability $\frac{1}{25}$. We also say that we generate non-isomorphic DAGs in the former case. Finally, when considering only transitive reductions, we discard the complete DAG. The probability to generate each of the remaining isomorphism classes (resp. labeled DAGs) with a uniform generator becomes $\frac{1}{5}$ (resp. $\frac{1}{19}$). This leads to four different uniformity definitions.

3.5 Scheduling

We consider a classic problem in parallel systems noted $P|p_j = 1, prec|C_{\max}$ in Graham's notation [GLLK79]. The objective consists in scheduling a set of tasks on homogeneous processors such as to minimize the overall completion time. The dependencies between tasks are represented by a precedence DAG (V, E) where $|V| = n$ is the number of tasks and $|E| = m$ the number of edges. Before starting its execution, all the predecessors of a task must complete their executions. The execution cost p_j of task j on any processor is unitary and there are no costs on the edges (i.e. no communication). A schedule defines on which processor and at which date each task starts executing such that no processor executes more than one task at any time and all precedence constraints are met. The problem consists in finding the schedule with the minimum makespan, i.e. overall completion time before the first task starting its execution and the last one completing its execution.

A possible schedule for the DAG of Figure 3.1(a) on two processors P_1 and P_2, assuming costs are unitary, consists in starting executing tasks 1 and 2 on processor P_1 as soon as possible (i.e. at times 0 and 1), while processor P_2 processes tasks 5, 4, 8, 7 and 3 similarly. The execution of task 6 follows the termination of task 2 on processor P_1 to satisfy the precedence constraint of task 7. The makespan of this schedule is 5.

This problem is strongly NP-hard [Ull75], while it is polynomial when there are no precedence constraints ($P|p_j = 1|C_{\max}$), which means the difficulty comes from the dependencies. Many polynomial heuristics have been proposed for this problem (see Section 5.2). With specific instances, such heuristics may be optimal. This is the case when the width does not exceed the number of processors, which leads to a potentially large length. Any task can thus start its execution as soon as it becomes available. The problem is also polynomial with the instance consisting of a chain of length len with $n - \text{len}$ additional independent tasks (i.e. $m = \text{len} - 1$ and the width equals $n - \text{len} + 1$, which is large when the length is small). In this case, any heuristic prioritizing critical tasks and scheduling all other tasks as soon as possible will be optimal.

Although this work studies random DAGs with heuristics for the specific problem $P|p_j = 1, prec|C_{\max}$, generated DAGs can be used for any scheduling problem with precedence constraints. While avoiding specific instances depending on their width and length is relevant for many scheduling problems, it is not necessary the case for all of them. For instance, with non-unitary processing costs, instances with large width and small length are difficult because the problem is strongly NP-Hard even in the absence of precedence constraints ($P||C_{\max}$) [GJ78].

The proposed mass measure has a direct implication in the scheduling problem $P|p_j = 1, prec|C_{\max}$. Consider a DAG $D = (V, E)$ whose minimum shape is 1; there exists a bottleneck vertex v such that the shape of the DAG is of the form $(X_1, \ldots, X_\ell, \{v\}, X_{\ell+1}, \ldots, X_k)$. The scheduling problem for D can be decomposed into two subproblems, one for the sub-DAG of D whose set of vertices is $\{v\} \cup \bigcup_{i \leq \ell} X_i$ and one for the sub-DAG of D whose set of vertices is $\{v\} \cup \bigcup_{i > \ell} X_i$. Using recursively this decomposition, the initial problem can be decomposed into $n_c + 1$ independent scheduling problems, where n_c is the number of bottleneck vertices.

Applying a brute force algorithm for the scheduling problems computes the optimal results in a time $T \leq n_c T_m$, where T_m is the maximum time required to solve the problem on a DAG with $\text{mass}^{\text{abs}}(D)$ vertices. Since exponential brute force exact approaches exist, it follows that if $\text{mass}^{\text{abs}}(D) = O(\log^k n)$ for a constant k, then an optimal solution of the scheduling problem can be computed in sub-exponential time. Consequently, scheduling heuristics are irrelevant for task graph with polylogarithmic absolute mass. Similarly, the same arguments work to claim that interesting instances for the scheduling problem must have quite a large absolute mass (not in $o(n)$). It is therefore preferable to have instances with no or few bottleneck vertices, that is a unitary mass.

The relevance of the mass property is limited to the class of scheduling problems for which the instance can be cut into independent subinstances. While the mass is still relevant with non-unitary processing costs, it is no longer the case when there are communication costs and requires to be adjusted.

The purpose of this work is first to identify such DAGs properties to determine how the uniform generation of DAGs should be constrained with the objective to control them to avoid easy instances. Second, we plan to analyze existing generation methods relatively to the identified properties.

Chapter 4

Analysis of DAGs Properties and Generation Methods

Contents

In this chapter, Section 4.1 is dedicated to analyzing properties of 8 special DAGs. In Section 4.2, we analyze 4 different existing generation methods of DAGs according to the selected properties in Section 4.1. Finally, we conclude this chapter in Section 4.3 by mentioning the most noteworthy obtained result.

4.1 Analysis of Special DAGs

To analyze the properties described in Section 3.2, we introduce in Table 4.1 a collection of special DAGs. The first three DAGs (D_{empty}, D_{complete} and D_{chain}) constitute extreme cases in terms of precedence. The next two DAGs ($D_{\text{out-tree}}$ and D_{comb}), to which we can add the reversal of the complete binary tree ($D_{\text{in-tree}} = D^R_{\text{out-tree}}$), are examples of binary tree DAGs. The last three DAGs ($D_{\text{bipartite}}$,

D_{square} and $D_{\text{triangular}}$) are denser with more edges and with a compromise between the length and the width for these last two DAGs.

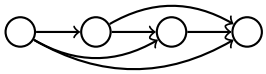
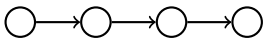
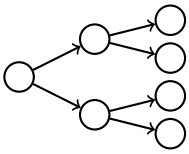
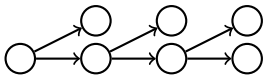
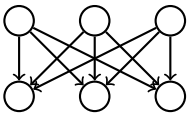
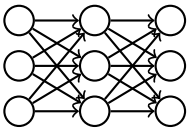
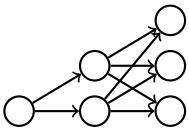
Name	description	representation
Empty (D_{empty})	no edge	
Complete (D_{complete})	maximum number of edges	
Chain (D_{chain})	transitive reduction of the complete DAG	
Complete binary tree ($D_{\text{out-tree}}$)	each non-leaf/non-root vertex has a unique predecessor and two successors	
Comb (D_{comb})	a chain where each non-leaf vertex has an additional leaf successor	
Complete bipartite ($D_{\text{bipartite}}$)	$\frac{n}{2}$ vertices connected to $\frac{n}{2}$ vertices	
Complete layer-by-layer square (D_{square})	similar to the complete bipartite with $\sqrt{n}$ layers of size $\sqrt{n}$	
Complete layer-by-layer triangular ($D_{\text{triangular}}$)	similar to the complete layer-by-layer square but the size of each new layer increases by 1	

Table 4.1: Special DAGs. The number of vertices n is assumed to be a power of two minus one for the tree, odd for the comb, even for the bipartite, a square for the square and a triangular number for the triangular (one of the form $1+2+3+\cdots+k$).

Table 4.2 illustrates the properties for these special DAGs. To discuss them, we analyze the most extreme values for each property. They are reached with the empty and complete DAGs except for the maximum standard deviations. Tables 4.3 and 4.4 synthesize the exact properties of the special DAGs. We publish all of these tables in our research report [CSH19].

The maximum value for the shape standard deviation is $\frac{n-1}{2}$ (reached with an empty DAG to which a single edge is added). When considering only transitive reductions (i.e. when discarding the complete DAG), the maximum value for the maximum degrees remains n with either a fork (a single source vertex is the

predecessor of all other vertices) or a join (the reversed fork). Proposition 4.1 states that the maximum number of edges among all transitive reductions is $\left\lfloor \frac{n^2}{4} \right\rfloor$ (reached with the bipartite DAG). As a corollary, the maximum value for the minimum and mean degrees is $\frac{n}{2}$. Studying the maximum achievable values for the degree standard deviations is left to future work.

Proposition 4.1. *The maximum number of edges among all transitive reductions of size n is $\left\lfloor \frac{n^2}{4} \right\rfloor$.*

Proof. Transitive reductions do not contain triangle (i.e. clique of size three), otherwise there is either a cycle or a redundant edge. By Mantel's Theorem [Man07], the maximum number of edges in a n-vertex triangle-free graph is $\left\lfloor \frac{n^2}{4} \right\rfloor$. This is the case for the complete bipartite DAG because the number of edges is $\frac{n^2}{4} = \left\lfloor \frac{n^2}{4} \right\rfloor$ when n is even and $\frac{n^2-1}{4} = \left\lfloor \frac{n^2}{4} \right\rfloor$ when n is odd. □

The edge-related properties are considerably affected when considering the transitive reduction of the complete DAG, i.e. the chain. Except for the standard deviations, all such properties are divided by $O(n)$. Considering transitive reductions can thus lead to different conclusions. The edge-related properties also highlight the asymmetry of both trees through the difference between input and output degrees.

Moreover, the density of a DAG appears to be quantified by the edge-related properties (e.g. the complete DAG and last three DAGs). Small values for the degree standard deviations characterize DAGs in which every vertex shares a similar structure (e.g. the empty DAG, chain, trees and combs). The length and shape-based properties show whether the DAG is short (empty and bipartite DAGs), balanced (the trees, square triangular DAGs) or long (the complete DAG, chain and combs). The maximum shape equals the width except for the reversed comb, which confirms the results shown in Table 3.1 on the similarity between the maximum shape and the width.

Finally, large values for the shape standard deviation characterize DAGs for which the parallelism varies significantly. This is the case for the trees and triangular DAG.

The analysis of these special DAGs provides some insight to select the relevant properties. Each given DAG possesses 18 properties, to which we add 9 properties by considering the transitive reduction and 7 properties by considering the reversal (for a total of 34 properties, n excluded). We limit the scope of our study to discard some properties for simplicity.

First, we assume that the generated DAGs are symmetrical and have similar properties through the reversal operation (which is the case for all special DAGs except for the trees and combs). This eliminates the 7 properties on the reversal. Moreover, the following properties become redundant with $\deg^{\max}$ and $\deg^{\text{sd}}$: $\deg^{\max}_{\text{in}}$, $\deg^{\max}_{\text{out}}$, $\deg^{\text{sd}}_{\text{in}}$ and $\deg^{\text{sd}}_{\text{out}}$ (which eliminates 8 additional properties). Second, we assume that only transitive reductions are meaningful in the context of scheduling without communication. This eliminates only 4 other properties because we keep the number of edges in the initial DAG as it provides meaningful information on the generation method. Moreover, we discard the mean degree because it is redundant with n and m, and provides little insight. Similarly, the minimum degree is not kept because it may be uninformative as it is low for source and sink vertices. We also discard the width and maximum shape because the mean shape provides a more global information. The mass already takes into account the minimum shape, which we discard. The last two shape properties (sh^1 and sh^{k}) provides only local information and are thus not kept.

This leaves 8 properties. In particular, we measure the following edge-related properties on the transitive reduction of any DAG: the number of edges, maximum degree and degree standard deviation. Additionally, we keep the length, the mean shape (even though it is redundant with n and len, it provides essential information on the global parallelism of the DAG), the shape standard deviation and the mass. The final property is the number of edges in the initial DAG.

Figure 4.1 shows the makespan obtained with three scheduling heuristics (described in Section 5.2) with all special DAGs as the number of processors varies. HEFT is always optimal because of the regularity of the DAG structures and because costs are unitary. This is also the case for the other heuristics most of the time. A zero mass, for long DAGs such as the complete DAG and chain, leads to an even easier scheduling problem where the number of processors has no impact. This confirms the discussion in Section 3.5 stating that low mass is characteristic of easy instances.

For the other DAGs, increasing the number of processors decreases the makespan until it reaches 1, 2, 7, 11, 15 and 50 for the empty DAG, bipartite DAG, trees, square DAG, triangular DAG and combs, respectively. Note that the stairs for the square are due to its layered structure. For the reversed comb, MinMin behaves poorly because this simple heuristic does not take into account the critical path and fill the processors with any of the initial source vertices.

Finally, the sub-optimal schedule produced by HCPT for the comb DAG is because, contrarily to HEFT with its insertion mechanism, this heuristic does not rely on backfilling and cannot schedule a task before any other already scheduled

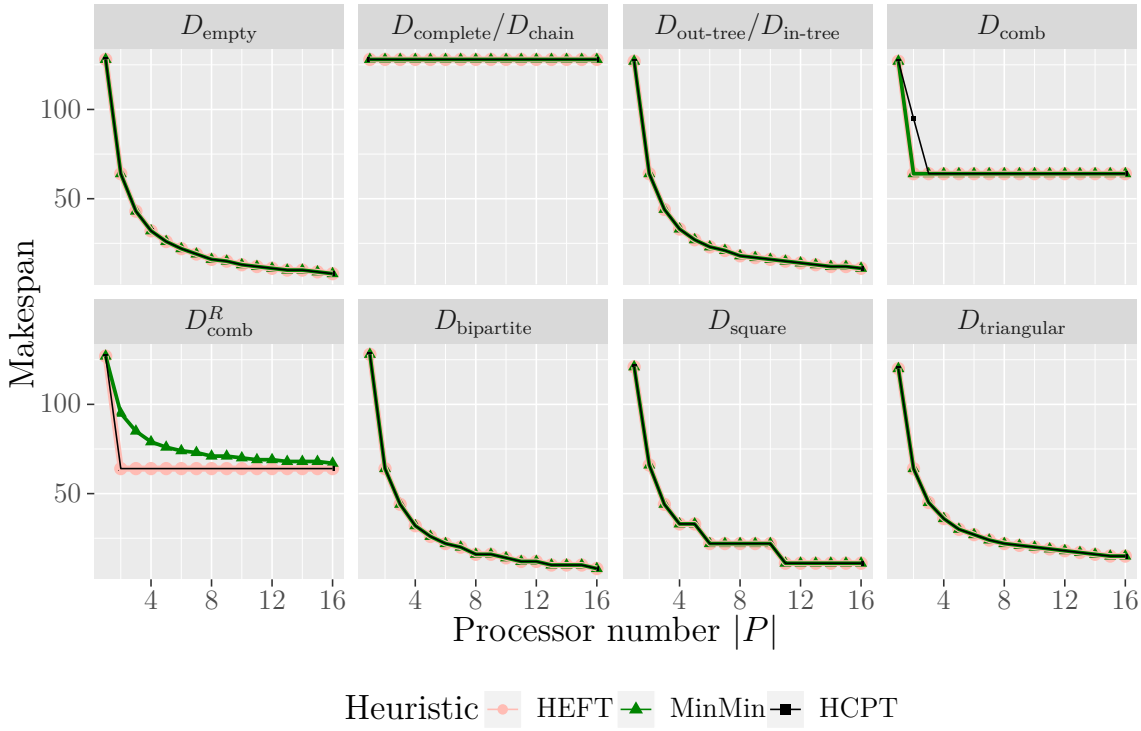


Figure 4.1: Makespan obtained with three heuristics (described in Section 5.2) on all special DAGs of Section 4.1 for $P|p_j = 1, prec|C_{\max}$. The number of vertices is $n = 128$ for the empty DAG, complete and bipartite DAGs, $n = 127$ for the trees and combs, $n = 121$ for the square DAG, and $n = 120$ for the triangular DAG $(1 + \cdots + 15 = 120)$.

tasks.

4.2 Analysis of Existing Generation Methods

This section covers and analyzes existing generation methods: the classic Erdős-Rényi algorithm; a uniform random generation method via a recursive approach; a poset-based method; and, an ad-hoc method frequently used in the scheduling literature.

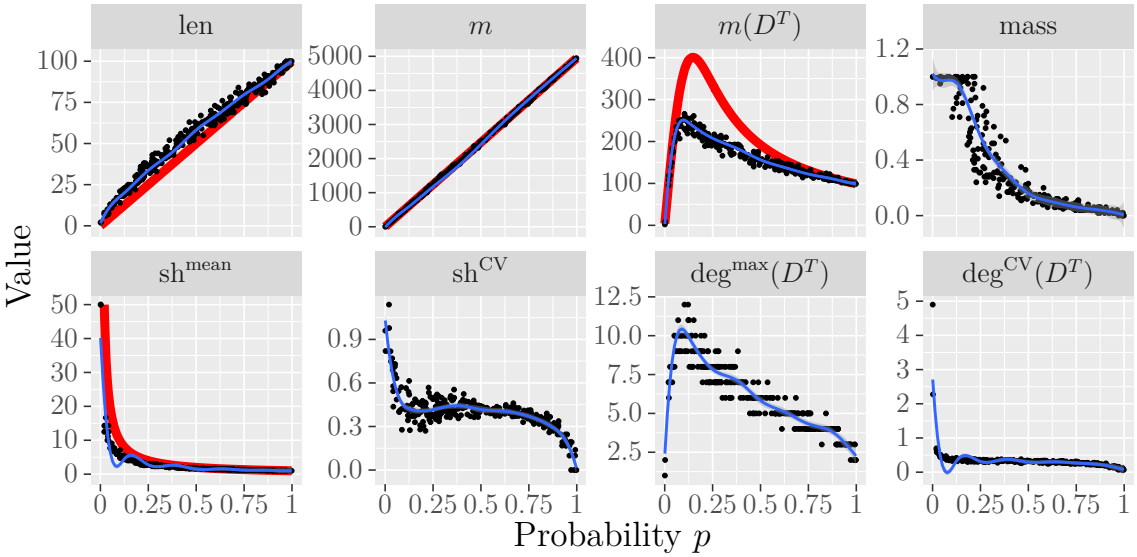


Figure 4.2: Properties of 300 DAGs of size $n = 100$ generated by the Erdős-Rényi algorithm with probability p uniformly drawn between 0 and 1. The smoothed line is obtained with a linear regression using a polynomial spline with 10 degrees of freedom. The degree CV (Coefficient of Variation) is the ratio of the mean degree to the degree standard deviation. Red lines correspond to formal results for the length and mean shape (Proposition 4.2), the number of edges, and the number of edges in the transitive reduction (Proposition 4.3).

4.2.1 Erdős-Rényi

This approach is based on the Erdős-Rényi algorithm [ER59] with parameter p (noted $G(n, p)$ in [Bol01]): an upper-triangular adjacency matrix is randomly generated. For each pair of vertices (i, j) with $i < j$, there is an edge from i to j with an independent probability p. The expected number of edges is therefore $p\frac{n(n-1)}{2}$.

The approach is not uniform (nor uniform up to isomorphism) for DAGs. For instance, a generator that is uniform up to isomorphism picks up the empty DAG with probability 1/6 (see Table 3.3). Moreover, a random generator that is uniform over all the DAGs (see Section 3.4 for the distinction) generates the empty DAG with probability 1/25. With $p = 0.5$, the Erdős-Rényi algorithm generates the DAG with no edges with probability 1/8.

Figures 4.2 and 4.3 show the effect of both parameters, probability p and size n, on the properties of the generated DAGs. For readability of both figures, each

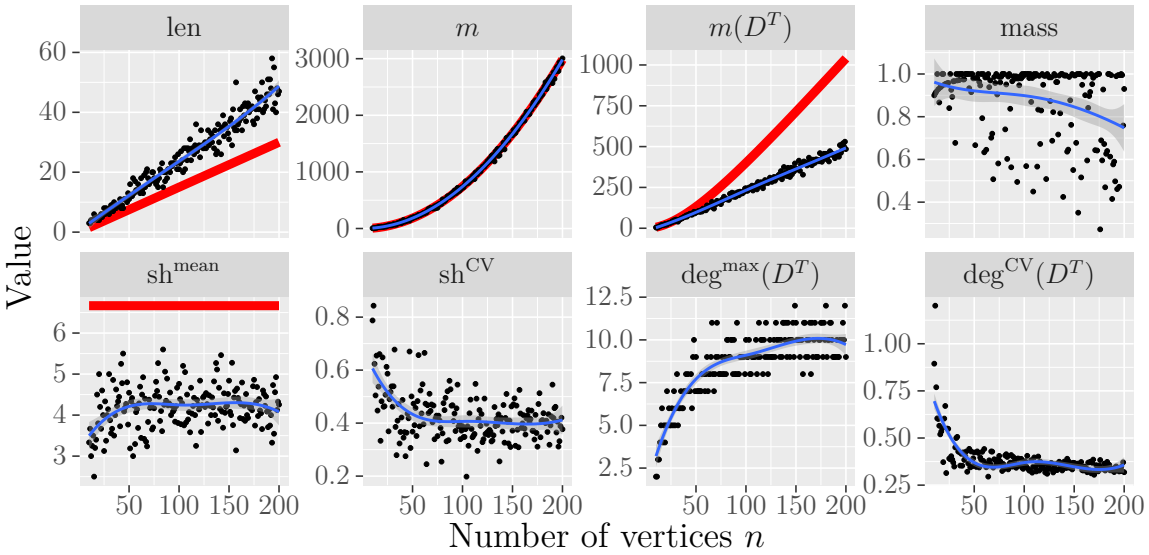


Figure 4.3: Properties of 191 DAGs generated by the Erdős-Rényi algorithm with probability $p = 0.15$ and for each size n between 10 and 200. The smoothed line is obtained with a linear regression using a polynomial spline with 4 degrees of freedom. Red lines correspond to formal results for the length and mean shape (Proposition 4.2), the number of edges, and the number of edges in the transitive reduction (Proposition 4.3).

standard deviation is replaced by a CV (Coefficient of Variation), which is the ratio of the standard deviation to the mean. The most evident effect on both figures is that the number of edges m increases linearly as p increases and quadratically as n increases, which is a direct consequence of the algorithm and the expected number of edges. Similarly, but with more variation, the length also increases as either parameter increases. This effect also concerns the mean shape because $\mathrm{sh}^{\mathrm{mean}} = \frac{n}{\mathrm{len}}$ (for instance, the length is close to 20 when $p = 0.125$, whereas the mean shape is close to 5). Therefore, on Figure 4.2, the mean shape decreases as the inverse function of the probability p because the length increases quasi-linearly with p. This effect is consistent with Proposition 4.2 which suggests that the expected mean shape is no greater than $\frac{1}{p}$.

Proposition 4.2. *Let D be a DAG with n vertices randomly generated by the Erdős-Rényi algorithm with parameter p. Denoting by $(X_1, \ldots, X_k)$ its shape decomposition, one has, for each $1 \leq i \leq k$, $\mathbb{E}(|X_i|) \leq \frac{1}{p}$.*

Proof. Let $M_{i,j}$ be the upper triangular matrix corresponding to D. Let $Y_i = \cup_{j<i} X_j$. If $j \in X_i$, then, for all $r < j$ such that $r \notin Y_i$, $M_{r,j} = 0$. Therefore, since the $M_{r,j}$ are independent Bernoulli random variables of parameter p, $\mathbb{P}(j \in X_i) \leq (1-p)^{j-|Y_i|}$. Consequently, $\mathbb{E}(x_i) \leq 1+(1-p)+\ldots+(1-p)^{n-|Y_i|} \leq \frac{1}{1-(1-p)} = \frac{1}{p}$. □

A more remarkable effect can be seen for the number of edges in the transitive reduction $m(D^T)$. This property shows that after a maximum around $p = 0.10$, adding more edges with higher probabilities leads to redundant dependencies and simplifies the structure of the DAG by making it longer. The same observation can be done with $\deg^{\max}(D^T)$. This is consistent with the fact that the algorithm generates the empty DAG when $p = 0$ and the complete DAG when $p = 1$. Proposition 4.3 also confirms this effect.

Proposition 4.3. *Let D be a DAG with n vertices randomly generated by the Erdős-Rényi Algorithm with parameter p. One has $\mathbb{E}(m(D^T)) \leq \frac{n-1}{p} - \frac{1-p^2}{p^3}(1-(1-p^2)^{n-1})$.*

Proof. Let $M_{i,j}$ be the upper triangular matrix corresponding to D.

$$A = \{(i,j) \mid 1 \leq i < j \leq n,\ (i,j) \in E \text{ s.t. } \forall i < r < j,\ (i,r) \notin E \text{ or } (r,j) \notin E\},$$

where E is the set of edges of D. By definition of D^T, if $(i,j) \in D^T$, then $(i,j) \in A$. Consequently,

$$m(D^T) \leq |A|. \tag{4.1}$$

Moreover, for every $i < j$, $\mathbb{P}((i,j) \in A) = \mathbb{P}(M_{i,j} = 1 \text{ and } \forall i < r < j,\ M_{i,r} = 0 \text{ or } M_{r,j} = 0)$. Since the $M_{i,j}$ are independent Bernoulli random variables,

$$\begin{aligned}\mathbb{P}((i,j) \in A) &= \mathbb{P}(M_{i,j} = 1)\Pi_{r=i+1}^{j-1}\mathbb{P}(M_{i,r} = 0 \text{ or } M_{r,j} = 0)\\ &= p\Pi_{r=i+1}^{j-1}(1 - \mathbb{P}(M_{i,r} = 1 \text{ and } M_{r,j} = 1))\\ &= p\Pi_{r=i+1}^{j-1}(1-p^2) = p(1-p^2)^{j-i-1}.\end{aligned}$$

Let $A_{i,j}$ be the Bernoulli random variable encoding that $(i,j) \in A$. One has $|A| = \sum_{i<j} A_{i,j}$. Consequently,

$$\begin{aligned}\mathbb{E}(|A|) &= \sum_{j=2}^{n}\sum_{i=1}^{j-1}\mathbb{E}(A_{i,j}) = \sum_{j=2}^{n}\sum_{i=1}^{j-1}p(1-p^2)^{j-i-1}\\ &= \sum_{j=2}^{n}\sum_{r=0}^{j-2}p(1-p^2)^{r} \quad \text{with } r = j-i-1\\ &= p\sum_{j=2}^{n}\frac{1-(1-p^2)^{j-1}}{1-(1-p^2)}\\ &= \frac{1}{p}\sum_{j=2}^{n}(1-(1-p^2)^{j-1})\\ &= \frac{n-1}{p} - \frac{1}{p}\sum_{j=2}^{n}(1-p^2)^{j-1}\\ &= \frac{n-1}{p} - \frac{1-p^2}{p}\sum_{j=0}^{n-2}(1-p^2)^{j}\\ &= \frac{n-1}{p} - \frac{1-p^2}{p}\frac{1-(1-p^2)^{n-1}}{1-(1-p^2)}\\ &= \frac{n-1}{p} - \frac{1-p^2}{p^3}(1-(1-p^2)^{n-1}).\end{aligned}$$

One can conclude using Equation (4.1). □

We rely on this apparent threshold around $p = 10\%$ to characterize three probability intervals: below 5%, between 5% and 15%, and above 15%. DAGs generated with a probability in the first interval are almost empty (hence a length lower than 10 and a mean shape higher than 10) with few vertices having some edges and many with no edges (hence the high degree standard deviation). For these DAGs, most edges are not redundant. Given the high shape standard deviation,

many tasks must be available at first. As mentioned in Section 3.5, these DAGs lead to a simplistic scheduling process that consists in starting each task on a critical path as soon as possible and then distributing a large number of independent tasks. Analogously, DAGs generated with probabilities p greater than 15% contain many edges that simplify the DAG structure by increasing the length and thus reducing the mean shape (recall that with a small width, the problem is easy, see Section 3.5). At the same time, the mass decreases continuously, allowing the problem to be divided into smaller problems. In particular, for probability p greater than 90%, DAGs are close to the chain, which is trivial to schedule. Therefore, most interesting DAGs are generated with probabilities between 5% and 15%.

As shown on Figure 4.3, the size of the DAG n has a simpler effect on the number of edges in the transitive reduction $m(D^T)$ than the probability p: $m(D^T)$ increases linearly with n (see Proposition 4.3). Moreover, the length increases with n as the shape mean remains constant (see Proposition 4.2). As a consequence, the mass decreases with n because the probability to obtain the value 1 increases in a vector with constant mean but increasing size. It is thus advisable to lower the probability with large sizes to maintain a constant mass.

The analysis of the Erdős-Rényi algorithm provides some insight on the desirable characteristics for the purpose of comparing scheduling heuristics. The effect of probability p illustrates the compromise between the length and mean shape to avoid simplistic instances that are easily tackled (see Section 3.5). Moreover, the maximum number of edges in the transitive reduction $m(D^T)$ is around $\frac{5}{2}n$ in both figures. However, we know that reaching $\frac{n^2}{4}$ is possible (Proposition 4.1) and layer-by-layer DAGs (square and triangular) are in $O(n^{\frac{3}{2}})$. Therefore, the Erdős-Rényi algorithm fails to generate DAGs with such large $m(D^T)$.

4.2.2 Uniform Random Generation

There are two main ways to provide a uniform random generator to uniformly generate elements of $\mathcal{D}^n$ (uniform over all labelled DAGs, see Section 3.4). The first one consists in using a classical recursive/counting approach [Rob73]. This counting approach relies on recursively counting the number of DAGs with a given number of source vertices, that is vertices with no in-going edges. See [KM15, Section 4] for a complete algorithm that uniformly generates random DAGs with this approach. The second one relies on MCMC approaches [MDB01, IC02, MP04]. We describe below the recursive approach.

Let $a_n = |\mathcal{D}^n|$, $a_{n,s}$ be the number of DAGs of $\mathcal{D}^n$ having exactly s source

vertices ($\mathrm{sh}^1 = s$). It is proved in [Rob73] that:

$$a_n = \sum_{k=1}^{n} a_{n,k} \quad \text{and} \quad a_{n,k} = \binom{n}{k} b_{n,k} \quad \text{with} \quad b_{n,k} = \sum_{s=1}^{n-k} (2^k - 1)^s 2^{k(n-k-s)} a_{n-k,s}.$$

First, we compute all values a_i and $a_{i,k}$ for $1 \leq i \leq n$ and $1 \leq k \leq i$ with the initial conditions $a_{i,i} = 1$ for $1 \leq i \leq n$. Next, a shape is generated using Algorithm 4.1 (RandomShape(n)), where $\oplus$ is the concatenation of vectors.

Algorithm 4.1: RandomShape(n)

Data: n, a_i, $a_{i,k}$ for $1 \leq i \leq n$ and $1 \leq k \leq i$.
Result: A shape with n elements.
1 Randomly generate $s \in \{1, n\}$ with distribution $\mathbb{P}(s = j) = \frac{a_{n,j}}{a_n}$;
2 **return** $[s] \oplus$ RandomShape($n - s$);

Finally, Algorithm 4.2 (ShapeToDAG($[s_1, \ldots, s_k]$)) builds the final DAG by using the shape vector. Each element of the shape vector represents a number of source vertices. The algorithm starts by generating the vertices of the first element of shape. Then, for the next element in the shape vector, it generates the vertices and randomly connects each one to a random vertex from the previous level. Then, it connects uniformly each vertex of the previous level to a vertex from the current level. Then, each vertex in the current level is connected to a vertex from other previous levels with probability $\frac{1}{2}$.

Figure 4.4 depicts the effect of the number of vertices on the selected DAG properties. Three effects are noteworthy: the length closely follows the function $\frac{3n}{2}$, the number of edges m is almost indistinguishable from the function $\frac{n^2}{4}$ and the number of edges in the transitive reduction $m(D^T)$ closely follows $1.4n$. The first effect is consistent with a theoretical result stating that the expected number of source vertices sh^1 in a uniform DAG is asymptotically 1.488 as $n \to \infty$ [Lis75]. This implies that the expected value for each shape element is close to this value by construction of the shape. Proposition 4.7 confirms this expectation is no larger than 2.25, which makes the DAG an easy instance for scheduling problems (see Section 3.5). For the second effect, we know that the average number of edges in a uniform DAG is indeed $\frac{n^2}{4}$ [MDB01, Theorem 2]. Despite the large amount of studies dedicated to formally analyzing uniform random DAGs, to the best of our knowledge, the last effect has not been formally considered. We finally observe that the mass decreases as the size n increases and it converges to zero when the size n tends to infinity (This result is proved below).

Algorithm 4.2: ShapeToDAG($[s_1, \ldots, s_k]$)

Data: $[s_1, \ldots, s_k]$ a shape with n elements.
Result: A DAG with n vertices.

for $i \in [1, \ldots, k]$ **do**
  for $j \in [1, \ldots, s_i]$ **do**
    Generate a vertex v with level i;
    if $i > 1$ **then**
      Connect a random vertex from level $i-1$ to vertex v;
      Connect each vertex from previous level to vertex v with probability $\frac{s_{i-1}\frac{2^{s_{i-1}-1}}{2^{s_{i-1}}-1}-1}{s_{i-1}-1}$;
      Connect any other vertex from other previous levels to vertex v with probability 0.5;
    end
  end
end
return *the resulting DAG;*

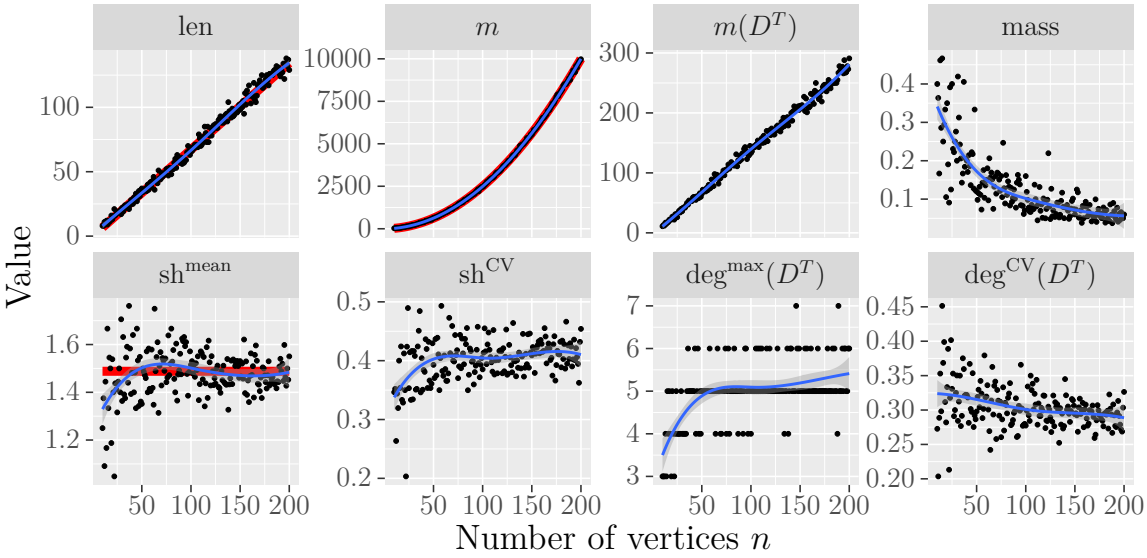


Figure 4.4: Properties of 191 DAGs generated by the recursive algorithm for each size n between 10 and 200. The smoothed line is obtained with a linear regression using a polynomial spline with 4 degrees of freedom. Red lines correspond to formal results for the length and mean shape, and the number of edges (the bound from Theorem 4.4 is discarded because it is too far).

As shown in Section 3.5, such instances can be decomposed into independent problems and efficiently solved with a brute force strategy. This leads to a sub-exponential generic time complexity with uniform instances. To obtain a similar average number of edges m with the Erdős-Rényi algorithm, we must choose a probability $p = 0.5$. We can compare both methods by considering $p = 0.5$ and $n = 100$ on Figures 4.2 and 4.4, respectively. We observe that DAGs generated by both methods share similar properties. This leads to similar conclusions as in Section 4.2.1.

Theorem 4.4. *Let D be a DAG uniformly and randomly generated among the labeled DAGs with n vertices. One has $\mathbb{P}(\text{mass}^{\text{abs}}(D) \geq \log^4(n)) \to 0$ when $n \to +\infty$.*

In order to prove Theorem 4.4, we will study probabilistic properties for DAGs.

Probabilistic properties The following represents the probabilistic properties of the shape of a DAG D randomly generated with the uniform distribution. In this context, we consider a random shape $(x_1, \ldots, x_k)$ generated by Algorithm 4.1 (RandomShape(n)), for a DAG with n vertices. Let $\mathbb{P}_\alpha$ be the probability of event parametrized by the number of vertices α (if we α is ommited, $\mathbb{P}$ denotes $\mathbb{P}_n$). Let $\mathbb{P}_\alpha$ be the probability of event parameterized by the number of vertices α (if α is ommited, $\mathbb{P}$ denotes $\mathbb{P}_n$). Note that the length k of the shape is a random variable and that the x_i's are dependent random variables. However, the distribution of x_i only depends on the sum s_i of the x_j's, with $j < i$. Formally, if $s_i < n$, $\mathbb{P}(x_i = r \mid x_1, \ldots, x_{i-1}) = \frac{a_{n-s_i,r}}{a_{n-s_i}} = \mathbb{P}(x_i = r \mid s_i)$ because the probability only depends on the value of s_i. It follows that if $s_i < n$, $\mathbb{P}(x_i = r \mid x_1, \ldots, x_{i-1}) = \mathbb{P}_{n-s_i}(x_1 = 1)$.

Therefore, using also [Lis75, Proposition 3]: if $n - s_i \geq 2$ and $r \geq 2$,

$$\mathbb{P}(x_i \leq r \mid s_i) \geq 1 - \frac{n - s_i - r}{n - s_i + 1} \frac{2}{(r+1)!\ 2^{\frac{r(r-1)}{2}}}. \tag{4.2}$$

It follows from Equation (4.2), for $r \geq 2$ and if $n - s_i \geq 2$,

$$\mathbb{P}(x_i > r \mid s_i) \leq \frac{n - s_i - r}{n - s_i + 1} \frac{2}{(r+1)!\ 2^{\frac{r(r-1)}{2}}} \leq \frac{2}{(r+1)!\ 2^{\frac{r(r-1)}{2}}}.$$

The above equation still holds if $n - s_i < 2$ since the probability is null.

These upper bounds show that the probability of having large values in the shape is very small. For instance, the probability that $x_i \geq 9$ is less than 10^{-11}.

Moreover, since for $r > 2$, $(r+1)! \geq 2^{r+1}$, one has for every $r > 2$,

$$\mathbb{P}(x_i > r \mid s_i) \leq 2^{-\frac{r(r+1)}{2}}.$$

It follows that

$$\mathbb{P}(x_i > r) \leq 2^{-\frac{r(r+1)}{2}}. \tag{4.3}$$

The following lemma will be useful.

Lemma 4.5. *One has, for every $r > 2$, $n \geq 2$, $\mathbb{P}(\max(x_i) > r) \leq n2^{-\frac{r(r+1)}{2}}$.*

Proof.

$$\begin{aligned}
\mathbb{P}(\max(x_i) > r) =& \mathbb{P}(\bigcup_{1 \leq i \leq k} \{x_i > r\}) \\
&\leq \sum_{i=1}^{k} \mathbb{P}(x_i > r) \\
&\leq \sum_{i=1}^{k} 2^{-\frac{r(r+1)}{2}} \quad \text{using Equation (4.3)} \\
&\leq n2^{-\frac{r(r+1)}{2}}
\end{aligned}$$

□

We can now claim an upper bound for the expected value of $\mathrm{sh}^{\max}(D) = \max(x_i)$.

Proposition 4.6. *One has $\mathbb{E}(\max(x_i)) = O(\log n)$.*

Proof. Let $h = \left\lfloor \sqrt{6 \log_2 n} \right\rfloor + 1$.

$$\begin{aligned}
\mathbb{E}(\max(x_i)) &= \sum_{r=1}^{n} \mathbb{P}(\max(x_i) = r).r \\
&= \sum_{r=1}^{h} r\mathbb{P}(\max(x_i) = r) \quad + \sum_{r=h+1}^{n} r\mathbb{P}(\max(x_i) = r) \\
&\leq \sum_{r=1}^{h} h\mathbb{P}(\max(x_i) = r) + \sum_{r=h+1}^{n} r\mathbb{P}(\max(x_i) = r) \\
&\leq h^2 + \sum_{r=h+1}^{n} r\mathbb{P}(\max(x_i) \geq r)
\end{aligned}$$

$$
\begin{aligned}
&\leq h^2 + \sum_{r=h+1}^{n} r\mathbb{P}(\max(x_i > r-1)\\
&\leq h^2 + \sum_{r=h+1}^{n} rn2^{-\frac{r(r-1)}{2}}\\
&\leq h^2 + n2^{-\frac{h(h+1)}{2}} \sum_{r=h+1}^{n} r\\
&\leq 6\log_2 n + n^3 2^{-\frac{h(h+1)}{2}}
\end{aligned}
$$

Since $r > 3$ for $n \geq 2$, we can apply Lemma 4.5 to eliminate the probability in the second term. Note that $2^{-\frac{h(h+1)}{2}} \leq 2^{-\frac{h^2}{2}} \leq 2^{-3\log_2 n}$. Since $n^3 2^{-3\log_2 n} = 1$, we have $\mathbb{E}(\max(x_i)) \leq 6\log_2 n + 1$, proving the result. □

It is proved in [Lis75] that $\mathbb{E}(x_1)$ converges to a constant (approximately 1.488) when n grows to infinity. One can easily obtain a bound for each level and each n.

Proposition 4.7. *One has, for every $n \geq 2$, every $1 \leq i \leq k$, $\mathbb{E}(x_i) \leq 2 + \frac{1}{4}$.*

Proof.

$$
\mathbb{E}(x_i) = \sum_{r=1}^{n} r\mathbb{P}(x_i = r) = \mathbb{P}(x_i = 1) + 2\mathbb{P}(x_i = 2) + \sum_{r=3}^{n} r\mathbb{P}(x_i = r)
$$

Note that $\mathbb{P}(x_i = 1) + \mathbb{P}(x_i = 2) \leq 1$.

$$
\begin{aligned}
\mathbb{E}(x_i) &\leq 2 + \sum_{r=3}^{n} \frac{r}{2^{r(r+1)/2}} \leq 2 + \sum_{r=3}^{n} \frac{1}{2^r} \frac{r}{2^{(r+1)/2}}\\
&\leq 2 + \sum_{r=3}^{n} \frac{1}{2^r}.
\end{aligned}
$$

Since $r \geq 3$ for $n \geq 2$, we can apply Lemma 4.5 to eliminate the probability in the last term. Since $\sum_{r=1}^{n} \frac{1}{2^r} \leq 1$, $\mathbb{E}(x_i) \leq 2 + \frac{1}{4}$. □

Previous results confirm experimental results in Section 4.2.2 and show that the values of the shape are all quite small. In order to evaluate the mass of a random DAG, we will now investigate the lengths of the bloc. More precisely, let

$$
\ell_{\max} = \max\{\ell \mid \exists i \text{ s. t. } (1-x_i)(1-x_{i+1})\dots(1-x_{i+\ell-1}) \neq 0\},
$$

the maximum length of a sequence of consecutive x_i non equal to 1.

It is proved in [Lis75, page 407] that for every $n > 0$, $\mathbb{P}(x_1 = 1) \geq \frac{1}{96} + \frac{17}{8(n+1)}$ in a uniform labeled DAG with n vertices and that $\mathbb{P}(x_1 = 1) \leq \frac{2}{3}$. Note that practical evaluation leads to claim that the probability to have only a single vertex in a level is greater than or equal to 1/3

Lemma 4.8. *For every $l > 0$, $\mathbb{P}(\ell_{\max} \geq \ell) \leq n\left(1 - \frac{1}{96}\right)^{\ell}$.*

Proof. Let $x_i = 0$ when $k < i \leq n$ (recall that k is the length of the shape vector). One has

$$\begin{aligned}\mathbb{P}(\ell_{\max} \geq \ell) &= \mathbb{P}(\cup_{i=1}^{n-\ell+1}\{x_i > 1, \ldots, x_{i+\ell-1} > 1\}) \\ &\leq \sum_{i=1}^{n-\ell+1} \mathbb{P}(x_i > 1, \ldots, x_{i+\ell-1} > 1)\end{aligned}$$

Now we claim that for every $0 < r \leq \ell$, every $i \leq n - \ell + 1$,

$$\mathbb{P}(x_{i+r} > 1 \mid x_i > 1, \ldots, x_{i+r-1} > 1) \leq 1 - \frac{1}{96} \tag{4.4}$$

Let $H_n = \{2, \ldots, n\}$. Each non zero x_i is in H_n. Therefore,

$$\begin{aligned}&\mathbb{P}(x_{i+r} > 1 \mid x_i > 1, \ldots, x_{i+r-1} > 1) = \\ &\qquad \sum_{\lambda \in H_n^r} \mathbb{P}(x_{i+r} > 1 \mid (x_i, \ldots, x_{i+r-1}) = \lambda)\mathbb{P}((x_i, \ldots, x_{i+r-1}) = \lambda).\end{aligned}$$

Now, $\mathbb{P}(x_{i+r} > 1 \mid (x_i, \ldots, x_{i+r-1}) = \lambda) = 0$ if $s_{i+r} \geq n$ (for $1 < i \leq n$, s_i is the sum $\sum_{j=1}^{i-1} x_i$). Moreover, if $s_{i+r} < n$, $\mathbb{P}(x_{i+r} > 1 \mid (x_i, \ldots, x_{i+r-1}) = \lambda) = \mathbb{P}_{n-s_{i+r}}(x_1 > 1) < (1 - \frac{1}{96})$ using Liskovets result (see also Page 67). Consequently,

$$\mathbb{P}(x_{i+r} > 1 \mid x_i > 1, \ldots, x_{i+r-1} > 1) \leq \left(1 - \frac{1}{96}\right) \sum_{\lambda \in H_n^r} \mathbb{P}((x_i, \ldots, x_{i+r-1}) = \lambda).$$

Since $\sum_{\lambda \in H_n^r} \mathbb{P}((x_i, \ldots, x_{i+r-1}) = \lambda) \leq 1$, Equation (4.4) holds.

Moreover, we show that $\mathbb{P}(x_i > 1) \leq 1 - \frac{1}{96}$ for $1 \leq i \leq n$. Liskovets result directly gives that $\mathbb{P}(x_1 > 1) < 1 - \frac{1}{96}$. For $1 < i \leq n$, $\mathbb{P}(x_i > 1) = \sum_{m=1}^{n} \mathbb{P}(s_i = m)\mathbb{P}(x_i > 1 | s_i = m) \leq 1 - \frac{1}{96}$ because $\mathbb{P}(x_i > 1 | s_i = m) = 1 - \mathbb{P}(x_i = 1 | s_i = m) - \mathbb{P}(x_i = 0 | s_i = m)$ and $\mathbb{P}(x_i = 1 | s_i = m) = \mathbb{P}_{n-m}(x_1 = 1) > \frac{1}{96}$.

Therefore,

$$\mathbb{P}(\ell_{\max} \geq \ell) \leq \sum_{i=1}^{n-\ell+1} \left(1 - \frac{1}{96}\right) \mathbb{P}(x_{i+1} > 1, \ldots, x_{i+\ell-1} > 1 | x_i > 1)$$

$$
\begin{aligned}
&\leq \sum_{i=1}^{n-\ell+1} \left(1-\frac{1}{96}\right)^2 \mathbb{P}(x_{i+2}>1,\ldots,x_{i+\ell-1}>1 | x_i>1, x_{i+1}>1) \\
&\leq \sum_{i=1}^{n-\ell+1} \left(1-\frac{1}{96}\right)^\ell \\
&\leq n\left(1-\frac{1}{96}\right)^\ell,
\end{aligned}
$$

proving the result. □

One can now prove Theorem 4.4.

Proof of Theorem 4.4. Consider the event $A_n = \mathrm{sh}^{\max}(D) \geq \log^2(n)$ and $B_n = \ell_{\max}(D) \geq \log^2(n)$. Using Markov Inequality and Proposition 4.6, $\mathbb{P}(A_n) \leq \frac{1}{\log n}$. Therefore, $\mathbb{P}(A_n) \to 0$ when $n \to +\infty$.

Moreover $\mathbb{P}(B_n) \leq n(1-\alpha_0)^{\log^2 n}$ by Lemma 4.8. But $\log(n(1-\alpha_0)^{\log^2 n}) = \log n + \log(1-\alpha_0)\log^2 n$. Since $0 < 1-\alpha_0 < 1$, $\log n + \log(1-\alpha_0)\log^2 n \to -\infty$ when $n \to +\infty$. Consequently $n(1-\alpha_0)^{\log^2 n} \to 0$ when $n \to +\infty$.

Therefore $\mathbb{P}(A_n \cup B_n) \to 0$ when $n \to +\infty$. □

4.2.3 Random Orders

The random orders method derives a DAG from randomly generated orders [Win85]. The first step consists in building K random permutations of n vertices. Each of these permutations represents a total order on the vertices, which is also a complete DAG with a random labeling. Intersecting these complete DAGs by keeping an edge iff it appears in all DAGs with the same direction leads to the final DAG. This is a variant of the algorithm presented in [CMP+10] where the transitive reduction in the last step is not performed because we already measure the properties on the transitive reduction.

Figure 4.5 shows the effect of the number of permutations K on the DAG properties with boxplots[1]. The extreme cases $K = 1$ and $K \to \infty$ are discarded from the figure for clarity. They correspond to the chain and the empty DAG, respectively. Recall that for the chain, $m(D^T) \approx \mathrm{len} = 100$, $m \approx 5{,}000$, $\mathrm{sh}^{\mathrm{mean}} = \deg^{\max}(D^T) = 1$ and the CVs and mass are zero. Similarly, for the empty DAG, len = mass = 1, the mean shape is 100 and all the other properties are zero.

[1] Each boxplot consists of a bold line for the median, a box for the quartiles, whiskers that extend at most to 1.5 times the interquartile range from the box and additional points for outliers.

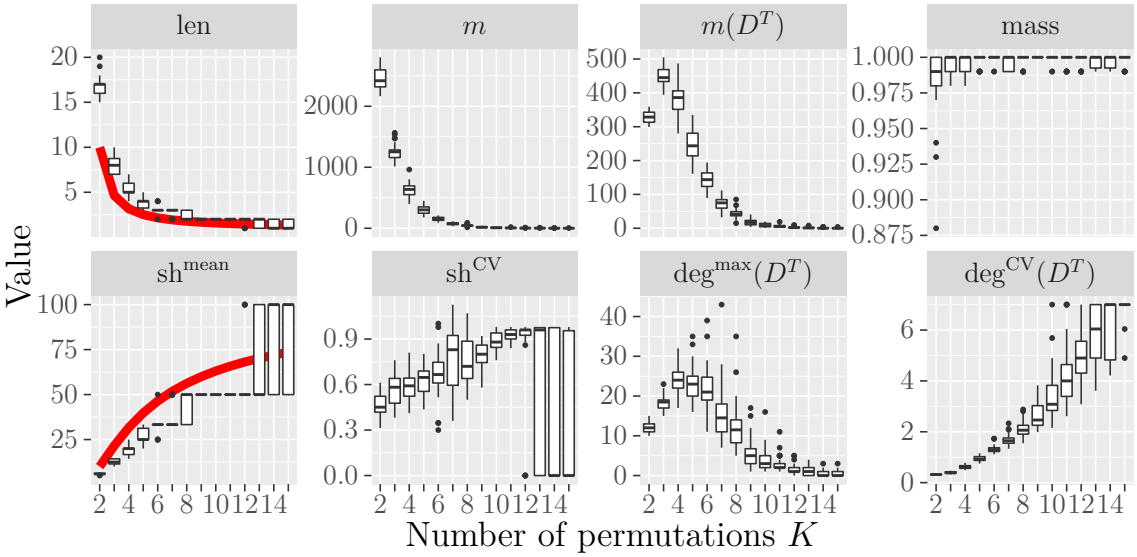


Figure 4.5: Properties of 420 DAGs of size $n = 100$ generated by the random orders algorithm for each number of permutations K between 2 and 15 (30 DAGs per boxplot). Red lines correspond to formal results for the length and mean shape.

The number of permutations quickly constrains the length. For instance, the length is already between 15 and 20 when $K = 2$ and at most 5 when $K \geq 5$. A formal analysis suggests that the length is almost surely in $O(n^{1/K})$ [Win85, Theorem 3], which is consistent with our observation. The number of edges and the maximum degree in the transitive reduction reach larger values than with previous approaches for any size n (twice larger than with the Erdős-Rényi algorithm). Moreover, the mass is always close to one for $K > 1$. Some specific values can finally be explained. First, the maximum value for $\deg^{\mathrm{CV}}(D^T)$ is exactly 7 and corresponds to DAGs of size $n = 100$ with a single edge (2 vertices have degree 1 and 98 others have 0). Also, the shape CV is at most 0.98 when the length is 2 (which frequently when $K \geq 10$). This CV corresponds to a shape with values 99 and 1.

Figure 4.6 shows the effect of the number of vertices n for a fixed number of permutations K. We selected $K = 3$ to have the maximum number of edges in the transitive reduction. The sublinear relation between the length and size n is again consistent with the previously cited result (i.e. $O(n^{1/K})$). Even though $K = 3$ is small, the length is already low, leading to line patterns for both the length and the mean shape. Note that the mass is frequently either 1 or almost 1 (i.e., $1 - \frac{1}{k}$), which corresponds to cases where only the last value of the shape sh^{k} is one.

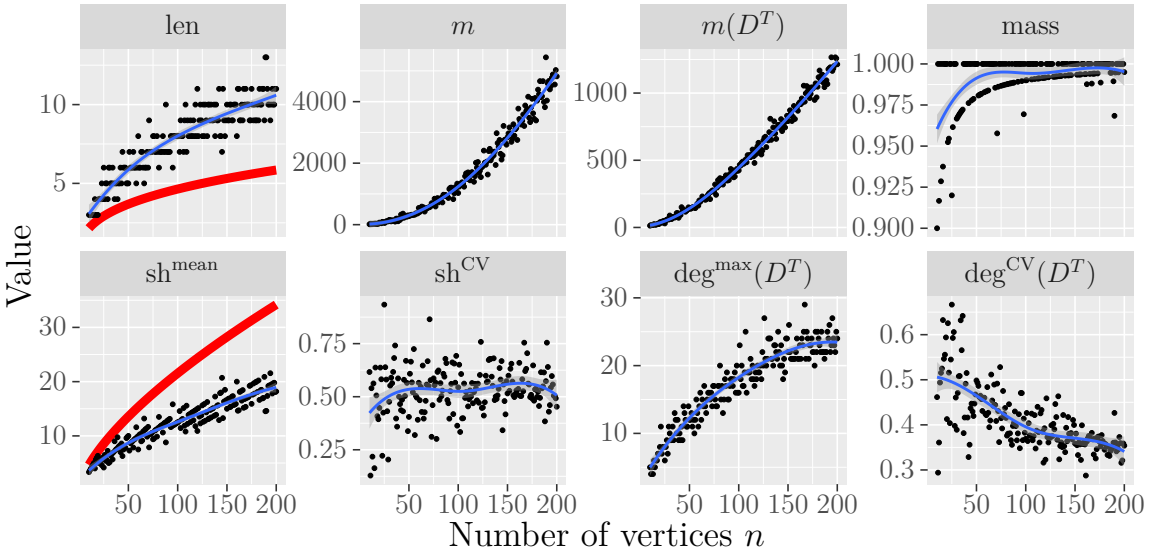


Figure 4.6: Properties of 191 DAGs generated by the random orders algorithm with $K = 3$ permutations and for each size n between 10 and 200. The smoothed line is obtained with a linear regression using a polynomial spline with 4 degrees of freedom. Red lines correspond to formal results for the length and mean shape.

The random orders method can generate denser DAGs than Erdős-Rényi or uniform DAGs without the mass issue, but with difficult control over the compromise between the length and the mean shape.

4.2.4 Layer-by-Layer

Many variants of the layer-by-layer principle have been used throughout the literature to assess scheduling algorithms and are covered in Section 3.3.3. This section analyzes the effect of three parameters (size n, number of layers k and connectivity probability p) using the following variant inspired from [CMP+10, GCJ17]. First, k vertices are affected to distinct layers to prevent any empty layer. Then, the remaining $n - k$ vertices are distributed to the layers using a balls into bins approach (i.e. a uniformly random layer is selected for each vertex). For each vertex not in the first layer, a random parent is selected among the vertices from the previous layer to ensure that the layer of any vertex equals its depth (similar to [DNSC09, GCJ17] and the recursive method in Section 4.2.2). Finally, random edges are added by connecting any pair of vertices from distinct layers from top to bottom with probability p.

This variant departs from [CMP+10, GCJ17] to ensure generated DAGs have a length equal to k and mean shape equal to n/k. Moreover, with some parameter values, this method produces some of the special DAGs covered in Section 4.1. It generates the empty DAG when $k = 1$, whereas it generates the complete DAG with $k = n$ and $p = 1$. To interpret the number of edges depicted in Figures 4.7 to 4.9, we study the case (called *regular*) when all layers have the same size n/k, which constitutes an approximation of the DAGs generated by the layer-by-layer variant studied in this section. When $p = 1$, the DAG is the bipartite one for $k = 2$ and the square one for $k = \sqrt{n}$. In such DAGs and when n is a multiple of k, the expected number of edges and the expected number of edges in the transitive reduction are given by Proposition 4.9.

Proposition 4.9. *In a regular layer-by-layer DAG with n vertices generated with parameters p and k, such that k divides n and all layers have the same size, the expected number of edges is*

$$\mathbb{E}(m) = n\left(1 - \frac{1}{k}\right)\left(p\left(\frac{n}{2} - 1\right) + 1\right) \tag{4.5}$$

and the expected number of edges in the transitive reduction is

$$\mathbb{E}(m(D^T)) \geq p(k-1)\left(\frac{n}{k}\right)^2 + (1-p)n\left(1 - \frac{1}{k}\right). \tag{4.6}$$

Proof. Let start with Equation 4.5. We analyze the average number of outgoing edges for a vertex from layer j with $1 \leq j < k$. By construction, there is at least one outgoing edge from this vertex to a successor of the next layer and there are $n(1 - \frac{j}{k}) - 1$ remaining potential successors. Thus, the average number of outgoing edges from this vertex is $p(n(1 - \frac{j}{k}) - 1) + 1$. It follows that the average number of outgoing edges from all vertices of layer j is $\frac{n}{k}(p(n(1 - \frac{j}{k}) - 1) + 1)$ because each layer is of size n/k. The average total number of edges is therefore:

$$\begin{aligned}
\sum_{j=1}^{k-1} \frac{n}{k}\left(p\left(n\left(1 - \frac{j}{k}\right) - 1\right) + 1\right) &= \sum_{j=1}^{k-1} p\frac{n^2}{k} - p\frac{n^2}{k^2}j - p\frac{n}{k} + \frac{n}{k} \\
&= (k-1)\left(p\frac{n^2}{k} - p\frac{n}{k} + \frac{n}{k}\right) - p\frac{n^2}{k^2}\sum_{j=1}^{k-1} j \\
&= n\left(1 - \frac{1}{k}\right)\left(pn - p + 1 - p\frac{n}{2}\right) \\
&= n\left(1 - \frac{1}{k}\right)\left(p\left(\frac{n}{2} - 1\right) + 1\right).
\end{aligned}$$

We follow a similar analysis for Equation 4.6. In the transitive reduction, any edge from two successive layers is never removed. We focus on the average number

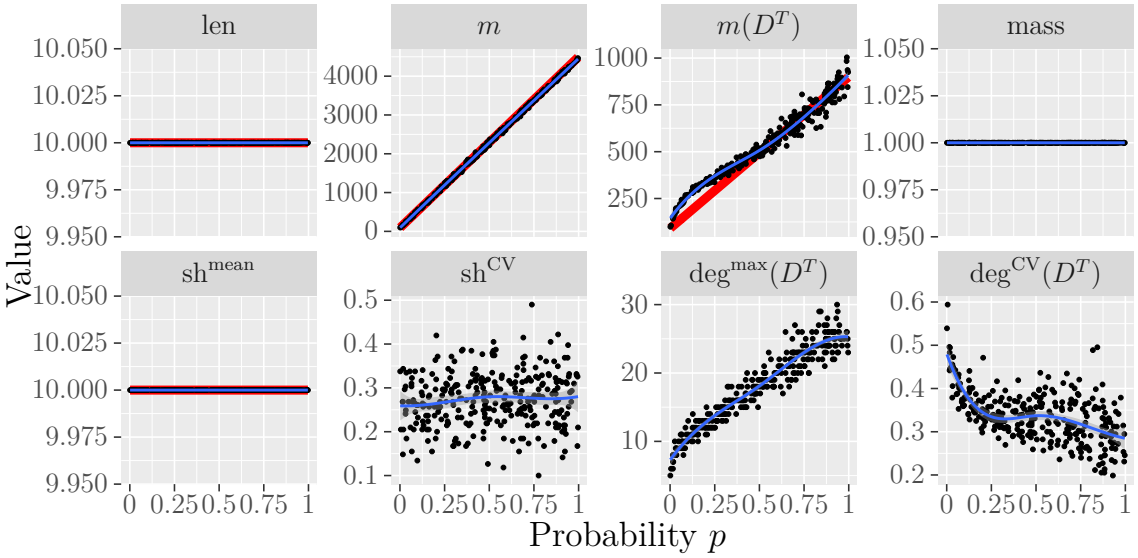


Figure 4.7: Properties of 300 DAGs of size $n = 100$ generated by the layer-by-layer algorithm with $k = 10$ layers and probability p uniformly drawn between 0 and 1. The smoothed line is obtained with a linear regression using a polynomial spline with 4 degrees of freedom. Red lines correspond to formal results for the length and mean shape, the number of edges (Equation 4.5), the number of edges in the transitive reduction (Equation 4.6), and the mass.

of such edges as a lower bound to the average total number of edges in the transitive reduction. By construction, there is again at least one outgoing edge from a vertex of any of the first $(k-1)$ layers and there are $\frac{n}{k}-1$ remaining potential successors in the next layer. It follows that the average number of outgoing edges from all vertices of any of the first $(k-1)$ layer is $\frac{n}{k}(p(\frac{n}{k}-1)+1)$. The average total number of edges between successive layers, a lower bound on the average number of edges in the transitive reduction, is therefore:

$$\begin{aligned}(k-1)\frac{n}{k}\left(p\left(\frac{n}{k}-1\right)+1\right) &= (k-1)p\frac{n^2}{k^2}-(k-1)p\frac{n}{k}+(k-1)\frac{n}{k}\\ &= p(k-1)\left(\frac{n}{k}\right)^2+(1-p)n\left(1-\frac{1}{k}\right).\end{aligned}$$

□

Figure 4.7 shows the effect of the probability p. The analysis for *regular* layer-by-layer DAGs closely approximates the results. The number of edges m is predicted to increase linearly from 90 to 4,500 (Equation 4.5), while this quantity in the

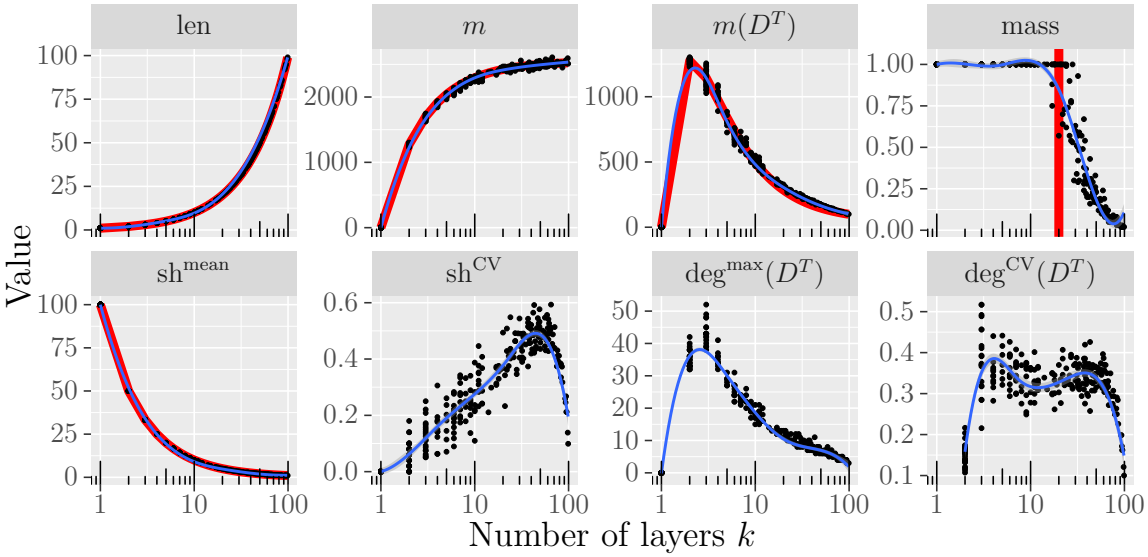


Figure 4.8: Properties of 300 DAGs of size $n = 100$ generated by the layer-by-layer algorithm with probability $p = 0.5$ and a number of layers k randomly drawn between 1 and 100 ($k = \lfloor e^{\mathcal{U}(\log(1),\log(101))} \rfloor$ where $\mathcal{U}(a, b)$ is a uniform distribution between a and b). The smoothed line is obtained with a linear regression using a polynomial spline with 5 degrees of freedom. Red lines correspond to formal results for the length and mean shape, the number of edges (Equation 4.5), the number of edges in the transitive reduction (Equation 4.6), and the mass.

transitive reduction $m(D^T)$ is expected to increase from 90 to 900 (Equation 4.6). Remark that this last property undergoes a steeper increase for probability $p < 0.1$ than for larger p. With many edges ($p > 0.1$), adding a new one is likely to result into the introduction of redundant edges, which is not the case for $p < 0.1$. More generally, the layered structure ensures a steady increase of $m(D^T)$ as the probability p increases because any edge between two consecutive layers cannot become redundant through the insertion of any edge. The mass is always close to one because the probability to have a layer with one vertex is close to zero with $k = 10$ layers.

Figure 4.8 represents the effect of the number of layers k. With *regular* layer-by-layer DAGs, the expected number of edges $\mathbb{E}(m)$ goes from 0 to 2,524.5 for $k = 1$ to 100 (Equation 4.5), which is close to the results with our layer-by-layer variant. The increase is steep because it is already 2,295 for $k = 10$, which is consistent with Figure 4.8. The number of edges in the transitive reduction $\mathbb{E}(m(D^T))$ decreases from an expected value of 1,275 to 99 as the number of layers goes from $k = 2$ to 100 (Equation 4.6). The expected value for $k = 10$ is 495 and is consistent with

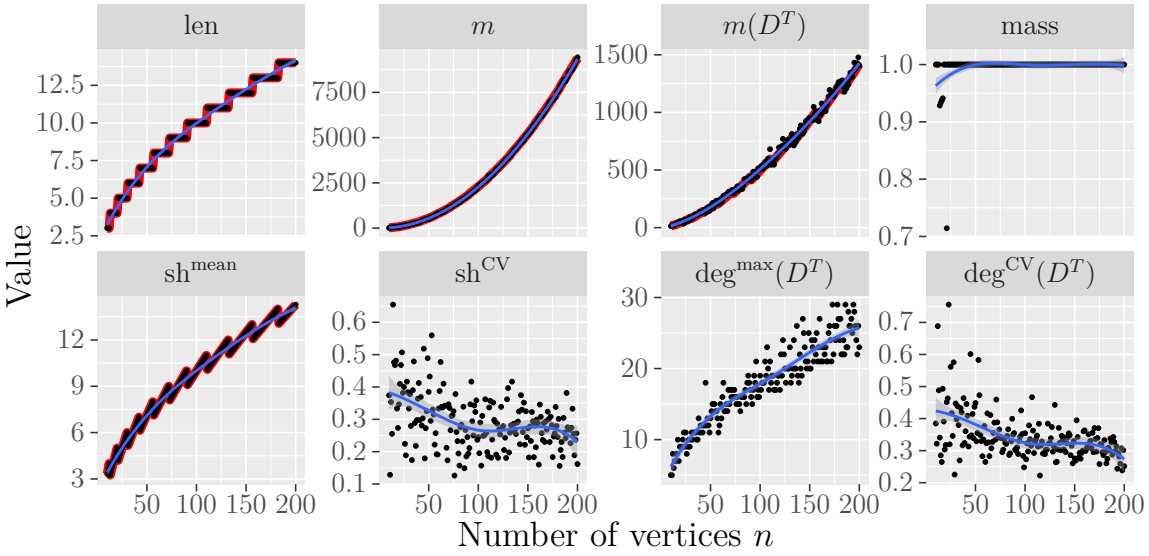


Figure 4.9: Properties of 191 DAGs generated by the layer-by-layer algorithm with probability $p = 0.5$, $k = \sqrt{n}$ (rounded to closest integer) layers and for each size n between 10 and 200. The smoothed line is obtained with a linear regression using a polynomial spline with 4 degrees of freedom. Red lines correspond to formal results for the length and mean shape, the number of edges (Equation 4.5), the number of edges in the transitive reduction (Equation 4.6), and the mass.

both Figures 4.7 and 4.8. Finally, the mass is unitary when there are at least two balls in each bin. Since there is initially one ball per bin, this occurs when there is at least one of the $n - k$ additional balls in each of the k bin. To compute if there are enough additional balls to have a unitary mass with probability greater than 0.5, we can use a bound for the coupon collector problem [LPW06, Proposition 2.4]. This occurs when $\lceil k \log(2k) \rceil + k < n$, which is the case for $k \leq 20$ when $n = 100$. This is consistent with Figure 4.8 where the mass becomes non-unitary around this value.

When varying the number of vertices n, we expect the number of edges m to increase quadratically from 20 to around 9,380 (Equation 4.5), which is consistent with the results on Figure 4.9. Similarly, the number of edges in the transitive reduction $m(D^T)$ is expected to increase quadratically from around 14.4 to around 1,420 (Equation 4.6).

In Figures 4.7 to 4.9, the length and mean shape show stable behavior consistent with our expectation. In all figures, the shape CV can formally be analyzed using the balls into bins model and we refer the interested reader to the specialized

literature [KSC78]. Finally, in the transitive reduction, the maximum degree $\deg^{\max}(D^T)$ has a similar trend as the number of edges $m(D^T)$.

To avoid non-unitary mass, the layer-by-layer method can be adapted to ensure that each layer has two vertices initially. For instance, we can rely on a uniform distribution between two and a maximum value, or on a balls into bins approach with two balls per bin initially. It is also possible to use the method described in Section 2.4 to have a uniform distribution of the vertices in the layers over all possible distributions and with a constraint on the minimum value.

4.3 Conclusion

In this chapter, 8 DAG properties have been selected among the proposed ones. It is noteworthy that the mass property leads to a sub-exponential generic time complexity for a class of scheduling problems when DAGs are generated uniformly at random. Finally, this chapter analyses, formally and empirically, existing random generation methods.

In the next chapter, we study how the generation methods impact the performance of scheduling heuristics with unitary costs.

DAG	m	$\deg^{\max}$	$\deg^{\max}_{\text{in}}$	$\deg^{\max}_{\text{out}}$	$\deg^{\min}$	$\deg^{\text{mean}}$	$\deg^{\text{sd}}$	$\deg^{\text{sd}}_{\text{in}}$	$\deg^{\text{sd}}_{\text{out}}$
D_{empty}	0	0	0	0	0	0	0	0	0
D_{complete}	$\frac{n^2}{2}$	n	n	n	n	n	0	$\frac{n}{\sqrt{12}}$	$\frac{n}{\sqrt{12}}$
D_{chain}	n	2	1	1	1	2	$\sqrt{\frac{2}{n}}$	$\frac{1}{\sqrt{n}}$	$\frac{1}{\sqrt{n}}$
$D_{\text{out-tree}}$ D_{comb}	n	3	1	2	1	2	1	$\frac{1}{\sqrt{n}}$	1
$D_{\text{in-tree}}$ D^R_{comb}	n	3	2	1	1	2	1	1	$\frac{1}{\sqrt{n}}$
$D_{\text{bipartite}}$	$\frac{n^2}{4}$	$\frac{n}{2}$	$\frac{n}{2}$	$\frac{n}{2}$	$\frac{n}{2}$	$\frac{n}{2}$	0	$\frac{n}{4}$	$\frac{n}{4}$
D_{square}	$n\sqrt{n}$	$2\sqrt{n}$	$\sqrt{n}$	$\sqrt{n}$	$\sqrt{n}$	$2\sqrt{n}$	$\sqrt{2\sqrt{n}}$	$\sqrt{\sqrt{n}}$	$\sqrt{\sqrt{n}}$
$D_{\text{triangular}}$	$\frac{2n\sqrt{2n}}{3}$	$2\sqrt{2n}$	$\sqrt{2n}$	$\sqrt{2n}$	2	$\frac{4}{3}\sqrt{2n}$	$\frac{2}{3}\sqrt{n}$	$\frac{\sqrt{n}}{3}$	$\frac{\sqrt{n}}{3}$

DAG	len	width	$\text{sh}^{\max}$	$\text{sh}^{\min}$	sh^{mean}	sh^{sd}	sh^{1}	sh^{k}	mass
D_{empty}	1	n	n	n	n	0	n	n	1
D_{complete} D_{chain}	n	1	1	1	1	0	1	1	0
$D_{\text{out-tree}}$	$\log_2(n)$	$\frac{n}{2}$	$\frac{n}{2}$	1	$\frac{n}{\log_2(n)}$	$\frac{n}{\sqrt{3\log_2(n)}}$	1	$\frac{n}{2}$	1
$D_{\text{in-tree}}$	$\log_2(n)$	$\frac{n}{2}$	$\frac{n}{2}$	1	$\frac{n}{\log_2(n)}$	$\frac{n}{\sqrt{3\log_2(n)}}$	$\frac{n}{2}$	1	1
D_{comb}	$\frac{n}{2}$	$\frac{n}{2}$	2	1	2	$\sqrt{\frac{2}{n}}$	1	2	1
D^R_{comb}	$\frac{n}{2}$	$\frac{n}{2}$	$\frac{n}{2}$	1	2	$\sqrt{\frac{n}{2}}$	$\frac{n}{2}$	1	$\frac{1}{2}$
$D_{\text{bipartite}}$	2	$\frac{n}{2}$	$\frac{n}{2}$	$\frac{n}{2}$	$\frac{n}{2}$	0	$\frac{n}{2}$	$\frac{n}{2}$	1
D_{square}	$\sqrt{n}$	$\sqrt{n}$	$\sqrt{n}$	$\sqrt{n}$	$\sqrt{n}$	0	$\sqrt{n}$	$\sqrt{n}$	1
$D_{\text{triangular}}$	$\sqrt{2n}$	$\sqrt{2n}$	$\sqrt{2n}$	1	$\sqrt{\frac{n}{2}}$	$\sqrt{\frac{n}{6}}$	1	$\sqrt{2n}$	1

Table 4.2: Approximate properties of special DAGs (negligible terms are discarded for clarity). More specifically, each approximate property approx(n) is related to the exact one exact(n) such that $\lim_{n\to\infty} \frac{\text{approx}(n)}{\text{exact}(n)} = 1$. The exact properties are given in Tables 4.3 and 4.4.

DAG	m	$\deg^{\max}$	$\deg^{\max}_{\text{in}}$	$\deg^{\max}_{\text{out}}$	$\deg^{\min}$	$\deg^{\text{mean}}$	$\deg^{\text{sd}}$	$\deg^{\text{sd}}_{\text{in}}$	$\deg^{\text{sd}}_{\text{out}}$
D_{empty}	0	0	0	0	0	0	0	0	0
D_{complete}	$\frac{n(n-1)}{2}$	$n-1$	$n-1$	$n-1$	$n-1$	$n-1$	0	$\sqrt{\frac{n^2-1}{12}}$	$\sqrt{\frac{n^2-1}{12}}$
D_{chain}	$n-1$	2	1	1	1	$2(1-\frac{1}{n})$	$\sqrt{\frac{2}{n}(1-\frac{2}{n})}$	$\sqrt{\frac{1}{n}(1-\frac{1}{n})}$	$\sqrt{\frac{1}{n}(1-\frac{1}{n})}$
$D_{\text{out-tree}}$ D_{comb}	$n-1$	3	1	2	1	$2(1-\frac{1}{n})$	$\sqrt{1-\frac{1}{n}-\frac{4}{n^2}}$	$\sqrt{\frac{1}{n}(1-\frac{1}{n})}$	$\frac{\sqrt{n-1}\sqrt{n+1}}{n}$
$D_{\text{in-tree}}$ D^R_{comb}	$n-1$	3	2	1	1	$2(1-\frac{1}{n})$	$\sqrt{1-\frac{1}{n}-\frac{4}{n^2}}$	$\frac{\sqrt{n-1}\sqrt{n+1}}{n}$	$\sqrt{\frac{1}{n}(1-\frac{1}{n})}$
$D_{\text{bipartite}}$	$\frac{n^2}{4}$	$\frac{n}{2}$	$\frac{n}{2}$	$\frac{n}{2}$	$\frac{n}{2}$	$\frac{n}{2}$	0	$\frac{n}{4}$	$\frac{n}{4}$
D_{square}	$n(\sqrt{n}-1)$	$2\sqrt{n}$	$\sqrt{n}$	$\sqrt{n}$	$\sqrt{n}$	$2(\sqrt{n}-1)$	$\sqrt{2\sqrt{n}-4}$	$\sqrt{\sqrt{n}-1}$	$\sqrt{\sqrt{n}-1}$
$D_{\text{triangular}}$	$\frac{k(k+1)(k-1)}{3}$	$2(k-1)$	$k-1$	k	2	$\frac{4}{3}(k-1)$	$\frac{\sqrt{2}}{3}(k-1)$	$\frac{\sqrt{(k-1)(k+2)}}{3\sqrt{2}}$	$\frac{\sqrt{(k-1)(k+14)}}{3\sqrt{2}}$

Table 4.3: Edge-related properties of special DAGs. For $D_{\text{triangular}}$, k is the length of the DAG ($k \approx \sqrt{2n} - \frac{1}{2}$ because $n = \frac{k(k+1)}{2}$).

DAG	len	width	$\mathrm{sh}^{\max}$	$\mathrm{sh}^{\min}$	$\mathrm{sh}^{\mathrm{mean}}$	$\mathrm{sh}^{\mathrm{sd}}$	sh^{1}	sh^{k}	mass
D_{empty}	1	n	n	n	n	0	n	n	1
D_{complete} D_{chain}	n	1	1	1	1	0	1	1	0
$D_{\text{out-tree}}$	$\log_2(n+1)$	$\frac{n+1}{2}$	$\frac{n+1}{2}$	1	$\frac{n}{\log_2(n+1)}$	$\sqrt{\frac{n}{\log_2(n+1)}\left(\frac{n+2}{3}-\frac{n}{\log_2(n+1)}\right)}$	1	$\frac{n+1}{2}$	$1-\frac{1}{n}$
$D_{\text{in-tree}}$	$\log_2(n+1)$	$\frac{n+1}{2}$	$\frac{n+1}{2}$	1	$\frac{n}{\log_2(n+1)}$	$\sqrt{\frac{n}{\log_2(n+1)}\left(\frac{n+2}{3}-\frac{n}{\log_2(n+1)}\right)}$	$\frac{n+1}{2}$	1	$1-\frac{1}{n}$
D_{comb}	$\frac{n+1}{2}$	$\frac{n+1}{2}$	2	1	$2(1-\frac{1}{n+1})$	$\sqrt{\frac{2}{n+1}(1-\frac{2}{n+1})}$	1	2	$1-\frac{1}{n}$
D^R_{comb}	$\frac{n+1}{2}$	$\frac{n+1}{2}$	$\frac{n+1}{2}$	1	$2(1-\frac{1}{n+1})$	$\frac{n-1}{n+1}\sqrt{\frac{n-1}{2}}$	$\frac{n+1}{2}$	1	$\frac{1}{2}+12n$
$D_{\text{bipartite}}$	2	$\frac{n}{2}$	$\frac{n}{2}$	$\frac{n}{2}$	$\frac{n}{2}$	0	$\frac{n}{2}$	$\frac{n}{2}$	1
D_{square}	$\sqrt{n}$	$\sqrt{n}$	$\sqrt{n}$	$\sqrt{n}$	$\sqrt{n}$	0	$\sqrt{n}$	$\sqrt{n}$	1
$D_{\text{triangular}}$	k	k	k	1	$\frac{k+1}{2}$	$\sqrt{\frac{k^2-1}{12}}$	1	k	$1-\frac{1}{n}$

Table 4.4: Vertex-related properties of special DAGs. For $D_{\text{triangular}}$, k is the length of the DAG ($k \approx \sqrt{2n} - \frac{1}{2}$ because $n = \frac{k(k+1)}{2}$).

Chapter 5

Performance Evaluation of Scheduling Algorithms for DAGs

Contents

Generating random task graphs allows the assessment of existing scheduling algorithms in different contexts. Numerous heuristics have been proposed for the problem denoted $P|p_j = 1, prec|C_{\max}$ (homogeneous tasks and processors, see Section 3.5) or generalizations of this problem. Such heuristics rely on different principles. Some simple strategies, like MinMin, executes available tasks on the processors that minimize completion time without considering precedence constraints. In contrast, many heuristics sort tasks by criticality and schedule them with the Earliest Finish Time (EFT) policy (e.g. HEFT and HCPT). Finally, other principles may be also used: migration for BSA [KA00], clustering for DSC [YG94], etc. We focus on the impact of generation methods on the performance of a selection of three heuristics for this problem: MinMin, HEFT, and HCPT.

5.1 Selected Scheduling Algorithms

HEFT [THW02a] (Heterogeneous Earliest Finish Time) first computes the upward rank of each task, which can be seen as a reverse depth (depth in the reversal DAG). It then considers tasks by decreasing order of their upward ranks and schedules them with the EFT policy. Backfilling is performed following an insertion policy that tries to insert a task at the earliest idle time between two already scheduled tasks on a processor if the slot is large enough to accommodate it. The time complexity of this approach is dominated by the insertion policy in $O(n^2)$. Numerous heuristics are equivalent to HEFT when tasks and processors are homogeneous: PEFT [AB14], HLEFT [ACD74], and HBMCT [SZ04].

HCPT [HJ03] (Heterogeneous Critical Parent Trees) starts by considering any task on a critical path by decreasing order of their depth. The objective is to prioritize the ancestors of such tasks and in particular when their depths are large. This process generates a priority list of tasks, and then schedules them with the EFT policy. The time complexity is $O(m+n\log(n)+n|P|)$ where $|P|$ is the number of processors.

Finally, MinMin [IK77a, Algorithm D] [FGA+98, minmin] considers all available tasks any time a processor becomes idle and execute available tasks on the processors that minimize completion time. With homogeneous tasks and processors, this algorithm is equivalent to MaxMin [IK77a, Algorithm E] [FGA+98, maxmin]. The time complexity is $O(m)$.

5.2 Performance of Scheduling Algorithms Regarding Generation Methods

Figure 5.1 shows the absolute difference between HEFT, HCPT and MinMin for each generation method covered in Section 4.2. We can notice that despite guaranteeing an unbiased generation, instances built with the recursive algorithm fail to discriminate heuristics except when there are two processors. Recall that the mean shape is close to 1.5 for such DAGs and few processors are sufficient to obtain a makespan equal to the DAG length (i.e. an optimal schedule).

In contrast, instances built with the random orders algorithm lead to different performance for each scheduling heuristic. However, this generation method has no uniformity guarantee and its discrete parameter K limits the diversity of generated DAGs.

Finally, the last two algorithms, Erdős-Rényi and layer-by-layer, fail to highlight a significant difference between MinMin and HEFT even though the former scheduling heuristic can be expected to be inferior to the latter as it discards the DAG structure.

To support these observations, we analyse below the maximum difference between the makespan obtained with HEFT and the ones obtained with the other two heuristics. Because it lacks any backfilling mechanism, HCPT performs worse than HEFT with an instance composed of the following two elements:

- First, a chain of length k with $|P| - 1$ additional tasks with predecessor the $(k-2)$th task of the chain and successor the kth task of the chain. Alternatively, this first element can be seen as a chain of length $k-3$ connected to a fork-join with width $|P|$.
- The second element is a chain of length $k-1$.

HCPT schedules the first element and then the second one afterward, leading to a makespan of $2k-1$ whereas the optimal one is k by reversing the execution order of the two elements. With $n = 100$ tasks and $|P| \leq 10$, the difference from HEFT with this instance is greater than or equal to 45.

Moreover, MinMin also performs worse with specific instances. Consider the ad-hoc instances considered in [CMSV18] each consisting of one chain of length k and a set of $k(|P|-1)$ independent tasks. Discarding the information about critical tasks prevents MinMin from prioritizing tasks from the chain. With $n = 100$ tasks and with $|P| \leq 10$, the worst-case absolute difference can be greater than or equals to 9 (when MinMin completes first the independent before starting the chain).

While the previously described difficult instances (with a large length and width) for HCPT rely on a specific weakness, it is interesting to analyze the properties of the difficult instances for MinMin. Each DAG is characterized by a length equals to $\text{len} = \frac{n}{|P|}$ and a number of edges in the transitive reduction $m(D^T) = \text{len} - 1$ (leading to a large width and a large shape standard deviation). With $n = 100$ tasks, with both HCPT and MinMin, the absolute difference from HEFT can be greater than or equal to nine.

5.3 Conclusion

We study, in Section 5.2, the impact of four generation methods of DAGs (Erdős-Rényi, uniform random generation, random orders, and layer-by-layer) on

three scheduling algorithms (HEFT, HCPT, MinMin). The experiments illustrate the need for better generation methods that control multiple properties while avoiding any generation bias.

In particular, this study highlights the need for a generator that uniformly samples all existing DAGs having a given properties: size n, number of edges m and/or $m(D^T)$, length and/or width, and with a unitary mass.

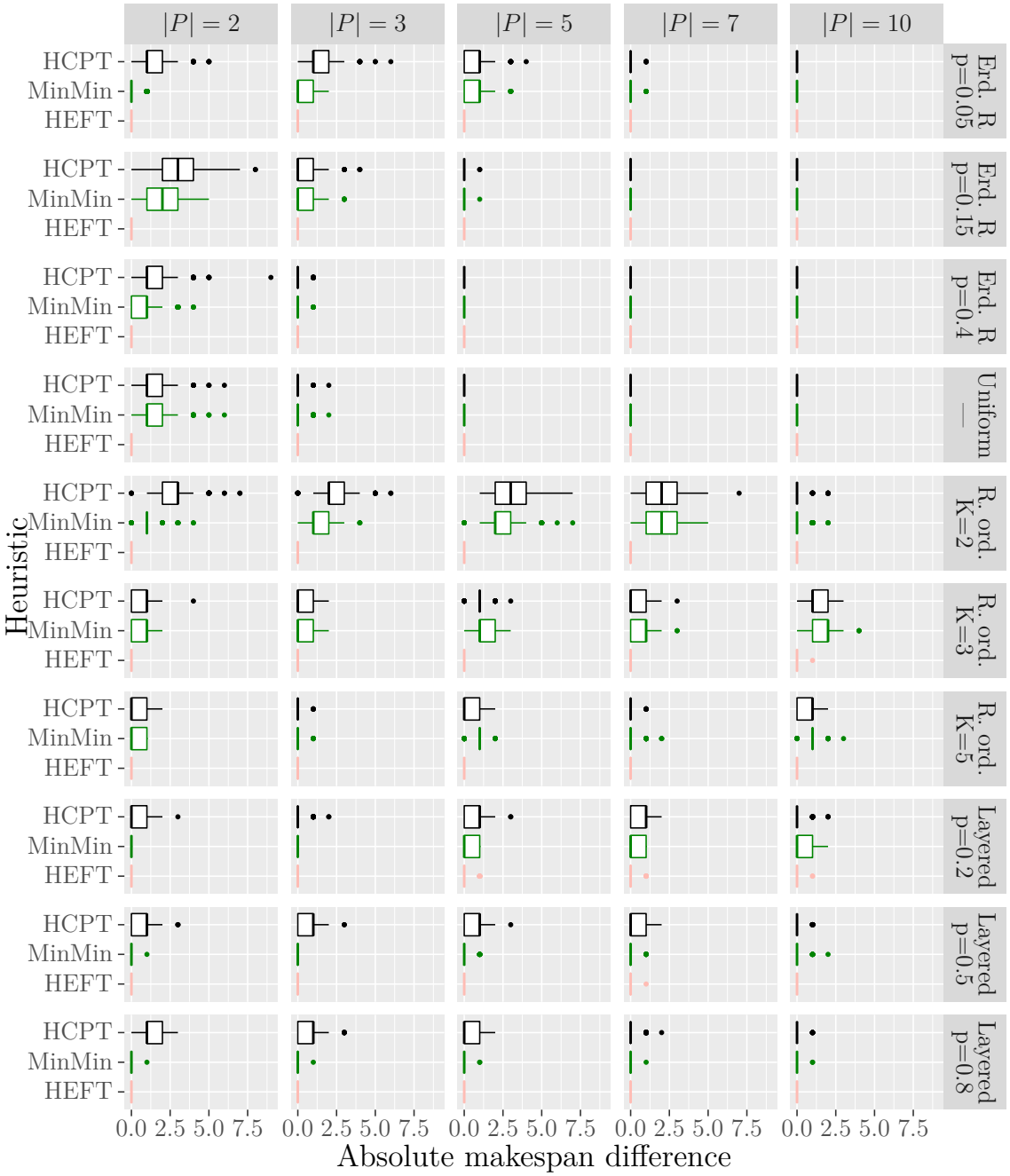


Figure 5.1: Difference between the makespan obtained with any heuristic and the best value among the three heuristics for each instance. Each boxplot represents the results for 300 DAGs of size $n = 100$ built with one of the following methods: the Erdős-Rényi algorithm (with probabilities 0.05, 0.15, and 0.4), the recursive algorithm, the random orders algorithm (with a number of permutations K equals to 2, 3, and 5), and the layer-by-layer algorithm (with a number of layers $k = 10$ and probabilities 0.2, 0.5, and 0.8). Costs are unitary and $|P|$ represents the number of processors.

Conclusion of Part II

Regarding the study of DAGs instances, our work contributes in three ways to the final objective of uniformly generating random DAGs belonging to a category of instances with desirable characteristics. At a first stage, we identify a list of 34 DAG properties and focus on a selection of 8 such properties. Among selected properties, the mass quantifies how much an instance can be decomposed into smaller ones. Then, we formally analyze and empirically assess existing random generation methods regarding the selected properties. We establish the sub-exponential generic time complexity for decomposable scheduling problems with uniform DAGs. Last, we study how the generation methods impact scheduling heuristics with unitary costs and we highlight the need for a generator that uniformly samples all existing DAGs having a given properties.

Part III

Cost matrices

Chapter 6

Motivation and Problem Statement for Cost Matrices

Contents

6.1 Introduction

As mentioned before, random generation is sensitive to bias when it relies on random instances with an uncontrolled distribution. For instance, among existing generation methods of cost matrices, the shuffling method [CP17] 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

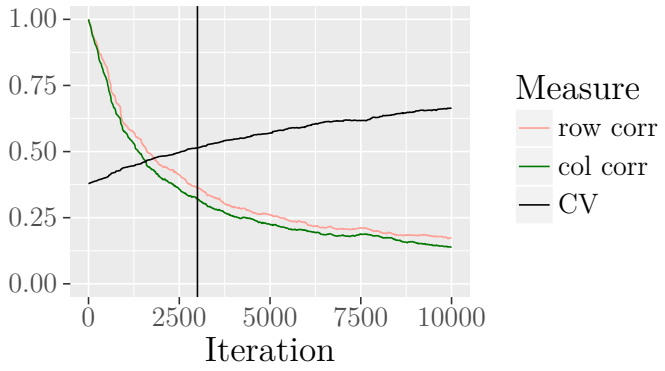


Figure 6.1: Cost Coefficient-of-Variation (ratio of standard deviation to mean) and mean row and column correlations at each iteration of the shuffling method [CP17] when generating a 100×30 cost matrix. The shuffling method arbitrarily stops after 3 000 iterations (represented by the black vertical line).

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 a difficult interpretability of results (see Figure 6.1).

In Section 6.2, we present cost matrices, their properties, and existing generation methods. In Section 6.3, we present the notion of contingency tables. We also show how we use the cost matrices as contingency tables and we describe the existing random generation methods of contingency tables. Last, we propose an asymptotic *uniform* random generation of cost matrices, which goes through two stages. The first stage consists in generating a pair of vectors that represent the fixed sum of the cost on each row and column of the cost matrix. The second stage is about generating, first, an initial cost matrix with elements bounded by the values of the pair of vectors generated in the first stage. Then, we generate randomly cost matrices by using the MCMC approach: at each iteration, we shuffle some values in the matrix, thus we obtain a new matrix. We move in the graph of the Markov Chain until being close to the stationary distribution. Finally, in Section 6.4, we define the studied problem in this part, which is ensuring a uniform distribution among the random generated cost matrices that have a given heterogeneity, which constrains the generation.

6.2 Cost Matrices

6.2.1 Definition

A cost matrix M is a matrix where the element of row i and column j, $M(i,j)$, represents the execution cost of task i on machine j.

Example. *The following matrix represents a cost matrix where rows represent tasks and columns represent machines* $M = \begin{array}{c} \\ T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \end{array} \begin{array}{c} \begin{array}{cc} M_1 & M_2 \end{array} \\ \begin{pmatrix} 5 & 2 \\ 3 & 4 \\ 4 & 4 \\ 3 & 1 \\ 2 & 3 \end{pmatrix} \end{array}$. *For example,* $M(5,2) = 3$ *is the execution cost of task 5 on machine 2.*

Note that it is possible to use rational values (instead of integers) by converting them to integers when generating randomly cost matrices. Practically, this convergence leads to large costs in the matrix, which causes a heavy calculation in terms of costs random generation.

6.2.2 Properties

Multiple criteria characterize cost matrices. In this thesis, we focus only on the study of the correlation and heterogeneity properties of cost matrices.

Armstrong [AJ97] was the first to introduce the concept of cost matrix heterogeneity. According to [AJ97], cost matrices are divided into four categories depending on their heterogeneity properties regarding tasks and machines: low/low, low/high, high/low, and high/high. For instance, high/low refers to high task heterogeneity and low machine heterogeneity. Task heterogeneity is the degree to which the execution times of different tasks vary for the same machine, i.e. the variation along the same column in the cost matrix. Analogously, machine heterogeneity is the degree to which the execution time of a given task varies for different machines, i.e. the variation along the same row of a cost matrix.

The correlation is a measure of the linear dependence between two variables. It has a value between +1, and -1, where +1 is total positive linear correlation, 0 is no linear correlation, and -1 is total negative linear correlation. In our context, the

correlation represents the dependence between rows, columns, of cost matrices. It also plays an important role as it corresponds to the machines being either related or specialized, with some affinity between the tasks and the machines [CHP17].

The heterogeneity and the correlation properties can be quantified using statistical measures. In the following, we mention the statistical measures used in our context where M represents a cost matrix with n rows and m columns:

- The cost Coefficient-of-Variation (CV): a statistical measure of dispersion of a probability distribution. It is the ratio of the standard deviation to the mean. The cost Coefficient-of-Variation is an indicator of the overall variance of the costs in the matrix.

$$\sqrt{\frac{1}{nm}\sum_{i=1}^{n}\sum_{j=1}^{m}\left(M(i,j)/\frac{N}{nm}-1\right)^2}$$

- The mean of row Coefficients-of-Variation (mean row CV): a statistical measure representing the mean of the Coefficient-of-Variation computed on each row of the cost matrix. It indicates the task heterogeneity, i.e. the variance of costs on the rows of the matrix.

$$\sum_{i=1}^{n}\frac{\sqrt{\frac{1}{m}\sum_{j=1}^{m}(M(i,j)-\frac{\overline{\mu}(i)}{m})^2}}{n\overline{\mu}(i)}$$

- The mean of column Coefficients-of-Variation (mean column CV): a statistical measure representing the mean of the Coefficient-of-Variation computed on each column of the cost matrix. It indicates the machine heterogeneity, i.e. the variance of costs on the columns of the matrix.

$$\sum_{j=1}^{m}\frac{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(M(i,j)-\frac{\overline{\nu}(j)}{n})^2}}{m\overline{\nu}(j)}$$

- The Pearson's χ^2 statistic: a statistical test for independence between categorical variables. In our context, the Pearson's χ^2 assesses the proportionality of the costs in the matrix.

$$\sum_{i=1}^{n}\frac{(M(i,j)-\overline{\mu}(i)\overline{\nu}(j)/N)^2}{\overline{\mu}(i)\overline{\nu}(j)/N}$$

- The mean of row correlations (mean row corr): as explained before, the correlation is a measure of the dependence between two variables. The mean of row correlations shows if rows of a cost matrix are proportional where $\rho(M(i,.),M(i',.))$ denotes the Pearson coefficient of correlation between rows i and i'.

$$\frac{1}{n(n-1)/2} \sum_{i=1}^{n-1} \sum_{i'=i+1}^{n} \rho(M(i,.), M(i',.))$$

- The mean of column correlations (mean col corr): it shows if columns of a cost matrix are proportional where $\rho(M(.,j), M(.j'))$ denotes the Pearson coefficient of correlation between columns j and j'.

$$\frac{1}{m(m-1)/2} \sum_{j=1}^{m-1} \sum_{j'=j+1}^{m} \rho(M(.,j), M(.,j'))$$

Table 6.1 provides a list of the most used notations in this part, which concerns cost matrices.

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. $\alpha \leq \overline{v}(i) \leq \beta$ 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 $\overline{\nu}$.
α, β	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 6.1: List of notations concerning the part of cost matrices.

6.2.3 Existing Generation Methods of Cost Matrices

Different methods of generating cost matrices exist. Some of these methods do not provide formal guarantee on the distribution of the generated instances, while other methods provide stronger formal guarantees. In the following, we briefly describe the existing methods and we mention their strengths and weaknesses.

Range-Based, Coefficient-of-Variation-Based Two main methods have been used in the literature: RB (range-based) and CVB (Coefficient-of-Variation-Based) [ASM+00, ASMH00]. 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. The produced cost matrices may not be representative of realistic settings [CHP17, CP17].

Shuffling-Based, Noise-Based More recently, two additional methods have been proposed for a better control of the heterogeneity: SB (shuffling-based) and NB (noise-based) [CP17]. 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: 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.

Correlation Noise-Based, Combination-Based Finally, CNB (correlation noise-based) and CB (combination-based) have been proposed to control the correlation [CHP17]. 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.

6.3 Cost Matrices as Contingency Tables

6.3.1 Contingency Tables

The term contingency table was first used in 1904 by Karl Pearson [Pea04]. Contingency tables (also called crosstabs or two-way tables) are an important data structure in statistics for representing the distribution of multivariate data. A contingency table is represented by a positive matrix with the sum of each row (resp. column) displayed in an additional total row (resp. column).

A contingency table M of n rows and m columns is the matrix $M(i,j)$ with row sums $\overline{\mu}$, and column sums $\overline{\nu}$. We denote by $\Omega^N_{n,m}(\overline{\mu},\overline{\nu})$ the set of $n \times m$ cotingency tables M over $\mathbb{N}$ such that for every $i \in \{1,\ldots,n\}$ and every $j \in \{1,\ldots,m\}$,

$$\sum_{k=1}^{m} M(i,k) = \overline{\mu}(i) \quad \text{and} \quad \sum_{k=1}^{n} M(k,j) = \overline{\nu}(j). \tag{6.1}$$

For example, the matrix

$$M_{\text{exa}} = \begin{pmatrix} 2 & 3 \\ 1 & 6 \\ 5 & 9 \end{pmatrix}$$

is in $\Omega_{3,2}(\overline{\mu}_{\text{exa}},\overline{\nu}_{\text{exa}})$, where $\overline{\mu}_{\text{exa}} = (5,7,14)$ and $\overline{\nu}_{\text{exa}} = (8,18)$.

Contingency tables are used for testing properties such as, for instance, the similarity between two rows or two columns [DG00a].

Example. *Let us consider that the number of registered students in the master's degree in the departement of computer science in a university is 100. The program offers 3 track options in the following areas: software engineering, networking and cryptology. The following matrix represents the contingency table showed in Table 6.2.*

$$\begin{array}{c} \\ 90 \\ 10 \end{array} \begin{array}{c} \begin{array}{ccc} 60 & 15 & 25 \end{array} \\ \begin{pmatrix} 55 & 12 & 23 \\ 5 & 3 & 2 \end{pmatrix} \end{array}$$

	Software Engineering (60)	Networking (15)	Cryptology (25)
Male (90)	55	12	23
Female (10)	5	3	2

Table 6.2: Example of contingency table representing the distribution of 100 registered students in the master's degree in the departement of computer science.

The first row represents males while the second one represents females. Columns represent tracks. The first column is for software engineering, the second one is for networking and the last one is for cryptology. For example, we see that the overall number of students in the track of software engineering (first column) is 60 of which 55 males and 5 females.

Since contingency tables are important data structures, we study in this work cost matrices as contingency tables, i.e. each cost matrix corresponds to a contingency table in order to control the execution costs of tasks and the machines speeds.

6.3.2 Existing Generation Methods of Contingency Tables

In the literature, MCMC is the most used approach to generate randomly contingency tables (see for example [CDHL05, DS+98, Fis12]). The first step is to generate a positive row vector r (resp. column c)representing the sums on rows and columns. Second, generate an initial positive contingency table (initial state of the MCMC) $M(i,j)$, where the sum of elements of row M_i and column M_j is equal to the element r_i (resp. c_j). To determine the next state $M'(i,j)$, we select, for instance, uniformly at random two distinct rows (resp. columns) from $M(i,j)$. Thus, we obtain a 2×2 submatrix $subM$ such as:

$$subM = \begin{pmatrix} M(i_1,j_1) & M(i_1,j_2) \\ M(i_2,j_1) & M(i_2,j_2) \end{pmatrix}.$$

Then, we add for instance $\begin{pmatrix} +1 & -1 \\ -1 & +1 \end{pmatrix}$ to $subM$ (taking into consideration that $subM$ remains positive). By making this change, we point out that $M'(i,j)$ differs from $M(i,j)$ in four entries. Moving enough in the Markov Chain leads to a contingency table that will almost be independent of the initial one.

6.3.3 Uniform MCMC Generation of Cost Matrices

Let n be the number of tasks, m the number of machines, and N the total sum of costs. To use MCMC with contingency tables, we need 2 vectors. Thus, the uniform random generation of cost matrices is based on two steps:

1. The first step in order to generate cost matrices 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 (average cost of each task) and column (average speed of each machine), which is related. Thus, we generate randomly and uniformly two vectors $\overline{\mu} \in \mathbb{N}^n$ and $\overline{\nu} \in \mathbb{N}^m$ satisfying:

$$\sum_{i=1}^{n} \overline{\mu}(i) = \sum_{j=1}^{m} \overline{\nu}(j) = N, \tag{6.2}$$

   This random generation can be performed uniformly using the recursive Algorithm 2.3 (Generate Sequences).

2. Next, the cost matrices can be generated using MCMC approach, and relying on contingency tables as explained in Section 6.3.2. From an initial contingency table (which represents the initial cost matrix), a random walk in the graph of contingency tables is performed. Recall that according to Theorem 2.10 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 graph of the Markov Chain leads to be in any state with the same probability.

The following example illustrates the uniform random generation of 2×3 cost matrices.

Example. *An illustration example of the uniformity of the random generation of cost matrices is depicted on Fig 6.2. This example is the same one described on Figure 2.7 where each state of the Markov Chain is represented by a cost matrix. The sum of each row is three and the sum of each column is two. As proved before, when the number of steps n grows, the distribution of cost matrices is close to a uniform distribution and the probability to be in each state tends to* $\frac{1}{7}$.

We see that we can generate uniformly at random both vectors, $\overline{\mu}$ representing the sums on rows (resp. $\overline{\nu}$ on columns) using the recursive Algorithm 2.3 (Generate Sequences). Then, based on these vectors, we can generate uniformly at random

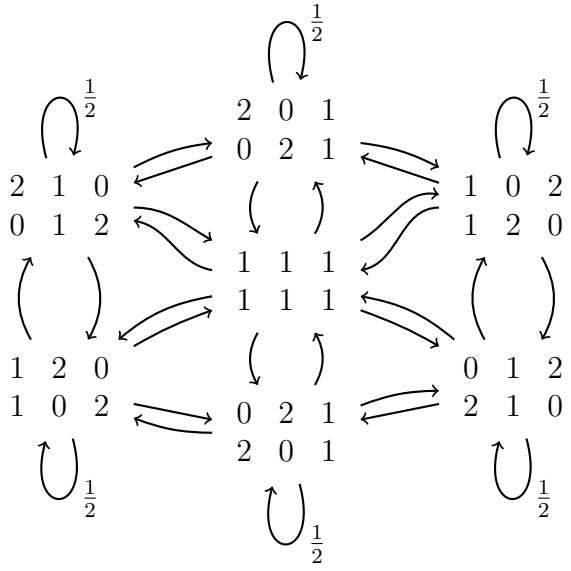


Figure 6.2: Example of the underlying graph of a Markov Chain when the sum of each row is 3 and the sum of each column is 2. Unless otherwise stated, each transition probability is $\frac{1}{6}$.

cost matrices using MCMC approach. But, we have to note that the distribution of the set of cost matrices generated using the vectors $\overline{\mu}$ and $\overline{\nu}$ (initial cost matrices) is not uniform. The following example will clarify this non-uniformity.

Example. *We want to generate a* 2×2 *matrix with an overall sum of elements* $N = 4$. *As we see in Table 6.3, there are 35 distinct matrices such that there is a unique matrix with all coefficients 1; four matrices with coefficients 0 and 4; six matrices with coefficients 0 and 2; and, twelve matrices with coefficients 0, 1 and 3; twelve matrices with coefficients 0, 1 and 2.*

The probability to sample uniformly at random any matrix from the distribution of the 35 matrices is $\frac{1}{35}$.

Now, to generate with our approach, using a contingency table, the matrix $\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$, *we have 5 different representations for a row and column sum vectors:*

$$\{(1,3),(3,1),(2,2),(4,0),(0,4)\}$$

The row and column sum vector representing the matrix $\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$ *is (2,2). Thus, the probability to sample the vector (2,2) over the set of the 5 different vectors is*

$\frac{1}{5}$. *Therefore, the probability to sample the row sum and the column sum vectors is* $\frac{1}{5} \times \frac{1}{5}$.

The probability to sample uniformly at random a matrix with a sum of 2 on each row and column is $\frac{1}{3}$, *since we have just 3 matrices with a sum of 2 on each row and column:* $\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$, $\begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}$, *and* $\begin{pmatrix} 0 & 2 \\ 2 & 0 \end{pmatrix}$.

Therefore, the probability to sample the matrix $\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$, *after sampling the row sum and column sum vectors (2,2), is* $\frac{1}{5} \times \frac{1}{5} \times \frac{1}{3} = \frac{1}{75}$, *which is not equal to the computed probability* ($\frac{1}{35}$) *for the uniform distribution. Consequently, we can notice that the distribution of the matrices generated using the vectors is not uniform.*

$\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$					
$\begin{pmatrix} 4 & 0 \\ 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 4 \\ 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 \\ 4 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 \\ 0 & 4 \end{pmatrix}$		
$\begin{pmatrix} 2 & 2 \\ 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 \\ 2 & 2 \end{pmatrix}$	$\begin{pmatrix} 0 & 2 \\ 0 & 2 \end{pmatrix}$	$\begin{pmatrix} 2 & 0 \\ 2 & 0 \end{pmatrix}$	$\begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}$	$\begin{pmatrix} 0 & 2 \\ 2 & 0 \end{pmatrix}$
$\begin{pmatrix} 3 & 1 \\ 0 & 0 \end{pmatrix}$ $\begin{pmatrix} 1 & 3 \\ 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 \\ 3 & 1 \end{pmatrix}$ $\begin{pmatrix} 0 & 0 \\ 1 & 3 \end{pmatrix}$	$\begin{pmatrix} 3 & 0 \\ 1 & 0 \end{pmatrix}$ $\begin{pmatrix} 1 & 0 \\ 3 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 3 \\ 0 & 1 \end{pmatrix}$ $\begin{pmatrix} 0 & 1 \\ 0 & 3 \end{pmatrix}$	$\begin{pmatrix} 3 & 0 \\ 0 & 1 \end{pmatrix}$ $\begin{pmatrix} 1 & 0 \\ 0 & 3 \end{pmatrix}$	$\begin{pmatrix} 0 & 3 \\ 1 & 0 \end{pmatrix}$ $\begin{pmatrix} 0 & 1 \\ 3 & 0 \end{pmatrix}$
$\begin{pmatrix} 2 & 1 \\ 1 & 0 \end{pmatrix}$ $\begin{pmatrix} 1 & 2 \\ 1 & 0 \end{pmatrix}$	$\begin{pmatrix} 2 & 1 \\ 0 & 1 \end{pmatrix}$ $\begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 2 & 0 \\ 1 & 1 \end{pmatrix}$ $\begin{pmatrix} 1 & 1 \\ 2 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 2 \\ 1 & 1 \end{pmatrix}$ $\begin{pmatrix} 1 & 1 \\ 0 & 2 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 \\ 2 & 1 \end{pmatrix}$ $\begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 \\ 1 & 2 \end{pmatrix}$ $\begin{pmatrix} 1 & 0 \\ 1 & 2 \end{pmatrix}$

Table 6.3: All 35 representations of the 4 coefficients for a 2×2 matrix with sum of elements $N = 4$.

6.4 Problem Statement and Contribution

As we see in Section 6.2, among existing random generators of cost matrices, we do not have methods with a strong formal guarantee on the distribution of the cost matrices with a control on their heterogeneity. Thus, it remains an open problem

to ensure a uniform distribution among the set of cost matrices that have a given task and machine heterogeneity.

Our objective is to control the row and column heterogeneity of cost matrices, while limiting the overall variance and ensuring a uniform distribution among the set of possible instances.

In order to control the heterogeneity, we show how to restrict this uniform random generation to interesting classes of vectors, and we extend known results for contingency tables to contingency tables with min/max constraints. We use MCMC to generate contingency tables with min/max constraints and we empirically evaluate the mixing time (the mixing time is the number of steps to walk in order to be close to the uniform distribution), using statistical estimations. Note that obtaining theoretical bound on mixing time is a very hard theoretical problem, still open in the general case of unconstrained contingency tables. Finally, we use our random generation process to evaluate the performance of different scheduling algorithms.

Chapter 7

Constrained Random Generation of Cost Matrices

Contents

In this chapter, Section 7.1 is dedicated to building symmetric and ergodic Markov Chains for the problem of generating uniformly at random cost matrices with min/max constraints. First, we define the set of all possible cost matrices with given row and column sums. Second, Markov Chains are proposed using a dedicated random mapping and are proved to be symmetric and ergodic. In Section 7.2, we use classical techniques to transform the Markov Chains into another symmetric ergodic MC mixing faster (i.e. the number of steps required to be close to the uniform distribution is smaller). In Section 7.3, we define three extremely different starting cost matrices. In Section 7.4, we graphically estimate the mixing time of the Markov Chain for different sizes of cost matrices. Then, in Section 7.5, we analyze the effect of the added constraints on the properties of the generated cost matrices. Finally, Section 7.6 is dedicated to concluding the chapter.

7.1 Symmetric Ergodic Markov Chain

As we show in Section 6.2.3, we can generate cost matrices by, first generating two vectors, then applying MCMC approach. In general, the graph of the Markov Chain is not explicitly built and neighborhood relation is defined by a random mapping on each state.

Constraining Contingency Tables We also introduce min/max constraints in order to control the variance of the value. We denote by $\Omega^N_{n,m}(\overline{\mu},\overline{\nu})$ the set of positive $n \times m$ matrices M over $\mathbb{N}$ as in Section 6.3.1.

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 such that $\alpha < \beta$. We denote by $\Omega^N_{n,m}(\overline{\mu},\overline{\nu})[\alpha,\beta]$ the subset of $\Omega^N_{n,m}(\overline{\mu},\overline{\nu})$ of matrices M such that for all i, j, $\alpha \leq M(i,j) \leq \beta$.

Consider the matrix

$$M_{\text{exa}} = \begin{pmatrix} 2 & 1 & 5 \\ 3 & 6 & 9 \end{pmatrix}.$$

For example, $M_{\text{exa}} \in \Omega_{2,3}(\overline{\mu}_{\text{exa}},\overline{\nu}_{\text{exa}})[0,10]$.

Moreover, according to Equation (6.1), one has

$$\begin{aligned} \Omega^N_{n,m}(\overline{\mu},\overline{\nu}) &= \Omega^N_{n,m}(\overline{\mu},\overline{\nu})[0,N] \\ &= \Omega^N_{n,m}(\overline{\mu},\overline{\nu})[0,\min(\max_{1\leq k\leq m}\overline{\mu}(k), \\ &\qquad \max_{1\leq k\leq n}\overline{\nu}(k))]. \end{aligned} \tag{7.1}$$

Now we consider min/max constraints on each row and each column. Let $\overline{\alpha}_c, \overline{\beta}_c \in \mathbb{N}^m$ and $\overline{\alpha}_r, \overline{\beta}_r \in \mathbb{N}^n$, such that for all $1 \leq i \leq n$, all $1 \leq j \leq m$, $\overline{\alpha}_r(i) < \overline{\beta}_r(i)$ and $\overline{\alpha}_c(j) < \overline{\beta}_c(j)$. We denote by $\Omega^N_{n,m}(\overline{\mu},\overline{\nu})[\overline{\alpha}_c,\overline{\beta}_c,\overline{\alpha}_r,\overline{\beta}_r]$ the subset of $\Omega^N_{n,m}(\overline{\mu},\overline{\nu})$ of matrices M satisfying: for all i, j, $\overline{\alpha}_c(j) \leq M(i,j) \leq \overline{\beta}_c(j)$ and $\overline{\alpha}_r(i) \leq M(i,j) \leq \overline{\beta}_r(i)$.

For instance,

$$M_{\text{exa}} \in \Omega_{2,3}(\overline{\mu}_{\text{exa}},\overline{\nu}_{\text{exa}})[(1,1,5),(4,8,9),(0,1),(5,9)].$$

Using Equation (6.1), one has for every $\alpha, \beta \in \mathbb{N}$,

$$\begin{aligned} \Omega^N_{n,m}(\overline{\mu},\overline{\nu})[\alpha,\beta] = \Omega^N_{n,m}(\overline{\mu},\overline{\nu})[&(\alpha,\ldots,\alpha), \\ &(\beta,\ldots,\beta),(\alpha,\ldots,\alpha),(\beta,\ldots,\beta)]. \end{aligned} \tag{7.2}$$

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 such that for all i, j $A_{\min}(i,j) < B_{\max}(i,j)$. We denote by $\Omega^N_{n,m}(\overline{\mu},\overline{\nu})[A_{\min}, B_{\max}]$ the subset of $\Omega^N_{n,m}(\overline{\mu},\overline{\nu})$ of matrices M such that for all i, j, $A_{\min}(i,j) \leq M(i,j) \leq B_{\max}(i,j)$.

For instance, one has $M_{\text{exa}} \in \Omega^N_{n,m}(\overline{\mu},\overline{\nu})[A_{\text{exa}}, B_{\text{exa}}]$, with

$$A_{\text{exa}} = \begin{pmatrix} 1 & 1 & 4 \\ 0 & 0 & 5 \end{pmatrix} \quad \text{and} \quad B_{\text{exa}} = \begin{pmatrix} 5 & 4 & 6 \\ 4 & 7 & 12 \end{pmatrix}.$$

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

$$\Omega^N_{n,m}(\overline{\mu},\overline{\nu})[\overline{\alpha}_c, \overline{\beta}_c, \overline{\alpha}_r, \overline{\beta}_r] = \Omega^N_{n,m}(\overline{\mu},\overline{\nu})[A, B], \tag{7.3}$$

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

Symmetry and Ergodicity of Markov Chains As explained before, the random generation process is based on symmetric ergodic Markov Chains. The following is dedicated to define such chains on state spaces of the form $\Omega^N_{n,m}(\overline{\mu},\overline{\nu})$, $\Omega^N_{n,m}(\overline{\mu},\overline{\nu})[\alpha,\beta]$, $\Omega^N_{n,m}(\overline{\mu},\overline{\nu})[\overline{\alpha}_c, \overline{\beta}_c, \overline{\alpha}_r, \overline{\beta}_r]$ and $\Omega^N_{n,m}(\overline{\mu},\overline{\nu})[A_{\min}, B_{\max}]$.

According to Equations (7.1), (7.2) and (7.3), it suffices to work on $\Omega^N_{n,m}(\overline{\mu},\overline{\nu})$ $[A_{\min}, B_{\max}]$. To simplify the notation, let us denote by Ω the set $\Omega^N_{n,m}(\overline{\mu},\overline{\nu})$ $[A_{\min}, B_{\max}]$.

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 [DC95] on $\Omega^N_{n,m}(\overline{\mu}, \overline{\nu})$. The difficulty is to prove that the underlying graph is strongly connected since the constraints are hindering the moves.

Theorem 7.1. *The Markov Chain $\mathcal{M}$ is symmetric and ergodic.*

The proof of Theorem 7.1 is based on Lemma 7.3 and 7.4.

Definition 7.2. *Let A and 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

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

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 7.3. *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 7.2. Moreover both r and s are odd. The sequence $u_s, \ldots, u_r$ satisfies the Conditions 2. to 5. of Definition 7.2. 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 7.2 and is therefore a stair sequence for A and B.

□

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 7.4. *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 7.3, 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 $A_{\min}(1,1) \leq A(1,1) - 1 < B_{\max}(1,1)$. Similarly, $A_{\min}(2,1) < A(2,1) + 1 \leq B_{\max}(2,1)$, $A_{\min}(1,2) < A(1,2) + 1 \leq B_{\max}(1,2)$ and $A_{\min}(2,2) \leq A(2,2) - 1 < B_{\max}(2,2)$. 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 Case 0 does not hold and if $A(1,\frac{r}{2}) < B_{\max}(1,\frac{r}{2})$, then, $k = 1$ 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:

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

Case 2: If Cases 0 to 1 do not hold. On one hand, $A(1,\frac{r}{2}) = B_{\max}(1,\frac{r}{2})$ because Case 1 does not hold. On the other hand, $r > 4$ because Case 0 does not hold.

Thus, $i_0 = \max\{i \mid 1 \leq i \leq \frac{r}{2} - 2 \text{ and } A(i, \frac{r}{2}) > A_{\min}(i, \frac{r}{2})\}$ exists (it is at least the case for $i = 1$ because $A(1, \frac{r}{2}) > A_{\min}(1, \frac{r}{2})$). In this case $t_1 = (i_0, i_0 + 1, i_0 + 1, \frac{r}{2}), t_2 = (i_0 + 1, i_0 + 2, i_0 + 2, \frac{r}{2}), \ldots, t_{\frac{r}{2}-1-i_0} = (\frac{r}{2} - 2, \frac{r}{2} - 1, \frac{r}{2} - 1, \frac{r}{2})$ works because $A(i, \frac{r}{2}) = A_{\min}(i, \frac{r}{2}) < B_{\max}(i, \frac{r}{2})$ for $i_0 < i \leq \frac{r}{2} - 2$. With $C = f(\ldots f(f(A, t_1), t_2) \ldots, t_{\frac{r}{2}-1-i_0})$, one has $d(C, B) = d(A, B) - 2 \times (\frac{r}{2} - i_0 - 1) - 2$ if $A(i_0, \frac{r}{2}) > B(i_0, \frac{r}{2})$, and $d(C, B) = d(A, B) - 2 \times (\frac{r}{2} - i_0 - 1)$ otherwise. Moreover, for every $\ell \leq \frac{r}{2} - 2 - i_0$, $f(\ldots f(f(A, t_1), t_2) \ldots, t_\ell) \in \Omega$. In this case, the following matrix is a difference matrix:

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

□

One can now prove Theorem 7.1.

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 $(A_k)_{k \geq 0}$ 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 7.4. □

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.

7.2 Rapidly Mixing Chains

The Markov Chain 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 which will not increase a lot the cost of a movement in the Markov Chain.

This approach, used for instance in [DG00b], allows moving faster, particularly for large N's.

Moving in $\Omega^N_{n,m}(\overline{\mu}, \overline{\nu})$, 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 $\Omega^N_{n,m}(\overline{\mu}, \overline{\nu})[\alpha, \beta]$, one has

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

This approach is used in the experiments described in Sections 7.4 and 7.5.

7.3 Initial Matrices Generation

The Markov Chain described in Section 7.1 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 $\Omega^N_{n,m}(\overline{\mu}, \overline{\nu})[A_{\min}, B_{\max}]$ 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 7.1 (Homogeneous Matrices))
- a *heterogeneous* matrix with largest cost CV (Algorithm 7.2 (Heterogeneous Matrices))
- a *proportional* matrix with smallest Pearson's χ^2 statistic (Algorithm 7.3 (Proportional Matrices))

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 7.1 (Homogeneous Matrices) to 7.3 (Proportional Matrices) 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 7.1 (Homogeneous Matrices) to 7.3 (Proportional Matrices) are used to study convergence without constraints $A_{\min}$ and $B_{\max}$. Only Algorithm 7.3 (Proportional Matrices) supports such constraints and is used to study their effects in Section 7.5.

Algorithm 7.1 (Homogeneous Matrices) 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 7.2 (Heterogeneous Matrices) 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 7.3 (Proportional Matrices) starts with the rounding of the rational proportional matrix (i.e. the matrix in which costs

Algorithm 7.1: Homogeneous Matrices

Input: Integer vectors $\overline{\mu}$, $\overline{\nu}$
Output: $M \in \Omega^N_{n,m}(\overline{\mu}, \overline{\nu})$

begin
  $M \leftarrow \{0\}_{1 \leq i \leq n, 1 \leq j \leq m}$
  while $\sum_{i=1}^{n} \sum_{j=1}^{n} M(i,j) \neq N$ **do**
    if $\max(\overline{\mu}(i))/m \geq \max(\overline{\nu}(j))/n$ **then**
      $i \leftarrow \arg\max_i \overline{\mu}(i)$
      sort $j_1, \ldots, j_m$ such that $\overline{\nu}(j_k) \leq \overline{\nu}(j_{k+1})$
      for $j_k \in \{j_1 \ldots, j_m\}$ **do**
        $d \leftarrow \min(\overline{\nu}(j_k), \frac{\overline{\mu}(i)}{m-k+1})$
        $M(i, j_k) \leftarrow M(i, j_k) + d$
        $\overline{\mu}(i) \leftarrow \overline{\mu}(i) - d$
        $\overline{\nu}(j_k) \leftarrow \overline{\nu}(j_k) - d$
    else
      perform the same operation on the transpose matrix (swapping $\overline{\mu}$ and $\overline{\nu}$)
  return M

Algorithm 7.2: Heterogeneous Matrices

Input: Integer vectors $\overline{\mu}$, $\overline{\nu}$
Output: $M \in \Omega^N_{n,m}(\overline{\mu}, \overline{\nu})$

begin
  $M \leftarrow \{0\}_{1 \leq i \leq n, 1 \leq j \leq m}$
  while $\sum_{i=1}^{n} \sum_{j=1}^{n} M(i,j) \neq N$ **do**
    $D \leftarrow \min(\overline{\mu}^T \cdot \mathbb{1}_m, \mathbb{1}_n^T \cdot \overline{\nu})$
    $i_{\max}, j_{\max} \leftarrow \arg\max_{i,j} D(i,j)$
    $d \leftarrow D(i_{\max}, j_{\max})$
    $M(i_{\max}, j_{\max}) \leftarrow d$
    $\overline{\mu}(i_{\max}) \leftarrow \overline{\mu}(i_{\max}) - d$
    $\overline{\nu}(j_{\max}) \leftarrow \overline{\nu}(j_{\max}) - d$
  return M

Algorithm 7.3: Proportional Matrices

Input: Integer vectors $\overline{\mu}$, $\overline{\nu}$, integer matrices $A_{\min}$, $B_{\max}$
Output: $M \in \Omega^N_{n,m}(\overline{\mu}, \overline{\nu})[A_{\min}, B_{\max}]$

begin
  $M \leftarrow \max(A_{\min}, \min(\lfloor \overline{\mu}^T \times \overline{\nu}/N + 1/2 \rfloor, B_{\max}))$
  $\overline{\mu}(i) \leftarrow \overline{\mu}(i) - \sum_{j=1}^{m} M(i,j)$ for $1 \leq i \leq n$
  $\overline{\nu}(j) \leftarrow \overline{\nu}(j) - \sum_{i=1}^{n} M(i,j)$ for $1 \leq j \leq m$
  while $\sum_{j=1}^{m} M(i,j) \neq \overline{\mu}(i)$ ***or*** $\sum_{i=1}^{n} M(i,j) \neq \overline{\nu}(j)$ **do**
    choose random i and j
    $d \leftarrow 0$
    if $M(i,j) < B_{\max}(i,j), (\overline{\mu}(i) > 0$ or $\overline{\nu}(j) > 0)$ **then** $d \leftarrow 1$
    if $M(i,j) > A_{\min}(i,j), (\overline{\mu}(i) < 0$ or $\overline{\nu}(j) < 0)$ **then** $d \leftarrow -1$
    $M(i,j) \leftarrow M(i,j) + d$
    $\overline{\mu}(i) \leftarrow \overline{\mu}(i) - d$
    $\overline{\nu}(j) \leftarrow \overline{\nu}(j) - d$
  return M

are proportional to the corresponding row and column costs) and proceeds to few random transformations to meet the constraints.

In Algorithms 7.1 (Homogeneous Matrices) and 7.2 (Heterogeneous Matrices), the argmin and argmax can return any index arbitrarily in case of several minimums. In Algorithm 7.2 (Heterogeneous Matrices), $\mathbb{1}_n$ denotes a vector of n ones. Finally, in Algorithms 7.2 (Heterogeneous Matrices) and 7.3 (Proportional Matrices), $\overline{\mu}^T$ denotes the transpose of $\overline{\mu}$, which is a column vector.

Example. *In this example, we illustrate 3 different cost matrices generated using Algorithms 7.1 (Homogeneous Matrices) to 7.3 (Proportional Matrices), where the vector (25, 35, 45, 55) represents the sum of each row and column.*

Homogeneous	*Heterogeneous*	*Proportional*
$\begin{pmatrix} 0 & 0 & 7 & 18 \\ 0 & 10 & 12 & 13 \\ 11 & 11 & 12 & 11 \\ 14 & 14 & 14 & 13 \end{pmatrix}$	$\begin{pmatrix} 25 & 0 & 0 & 0 \\ 0 & 35 & 0 & 0 \\ 0 & 0 & 45 & 0 \\ 0 & 0 & 0 & 55 \end{pmatrix}$	$\begin{pmatrix} 4 & 5 & 7 & 9 \\ 5 & 8 & 10 & 12 \\ 7 & 10 & 13 & 15 \\ 9 & 12 & 15 & 19 \end{pmatrix}$

7.4 Mixing Time Estimation

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 $t_{\text{mix}}(\varepsilon)$ of an ergodic Markov Chain is the number of steps required in order to be ε-close to the stationary distribution(see 2.13). Computing theoretical bounds on mixing time is a hard theoretical problem. For two-rowed contingency tables, $t_{\text{mix}}(\varepsilon)$ is in $O(n^2 \log(\frac{N}{\varepsilon}))$ [DG00b] and it is conjectured that it is in $\Theta(n^2 \log(\frac{n}{\varepsilon}))$. The results are extended and improved in [CDG+06] 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 a 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 properties in order to observe the convergence. For our work, used parameters are defined in Section 6.2.2.

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 7.3, we monitor the measures defined in Section 6.2.2 in order to observe the convergence (here, approximately after 6 000 iterations). It is, for instance, depicted in Figure 7.1 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[1] for several walking steps, as in Figure 7.2 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.

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 7.1. A quick analysis that is not provided in this manuscript shows that the mixing time seems to be linearly bounded by $nm \log^3(nm)$. We did not prove this mixing time, this is only an observation and may depends on constraints and sampling rejection.

[1] Each boxplot consists of a bold line for the median, a box for the quartiles, whiskers that extend at most to 1.5 times the interquartile range from the box and additional points for outliers.

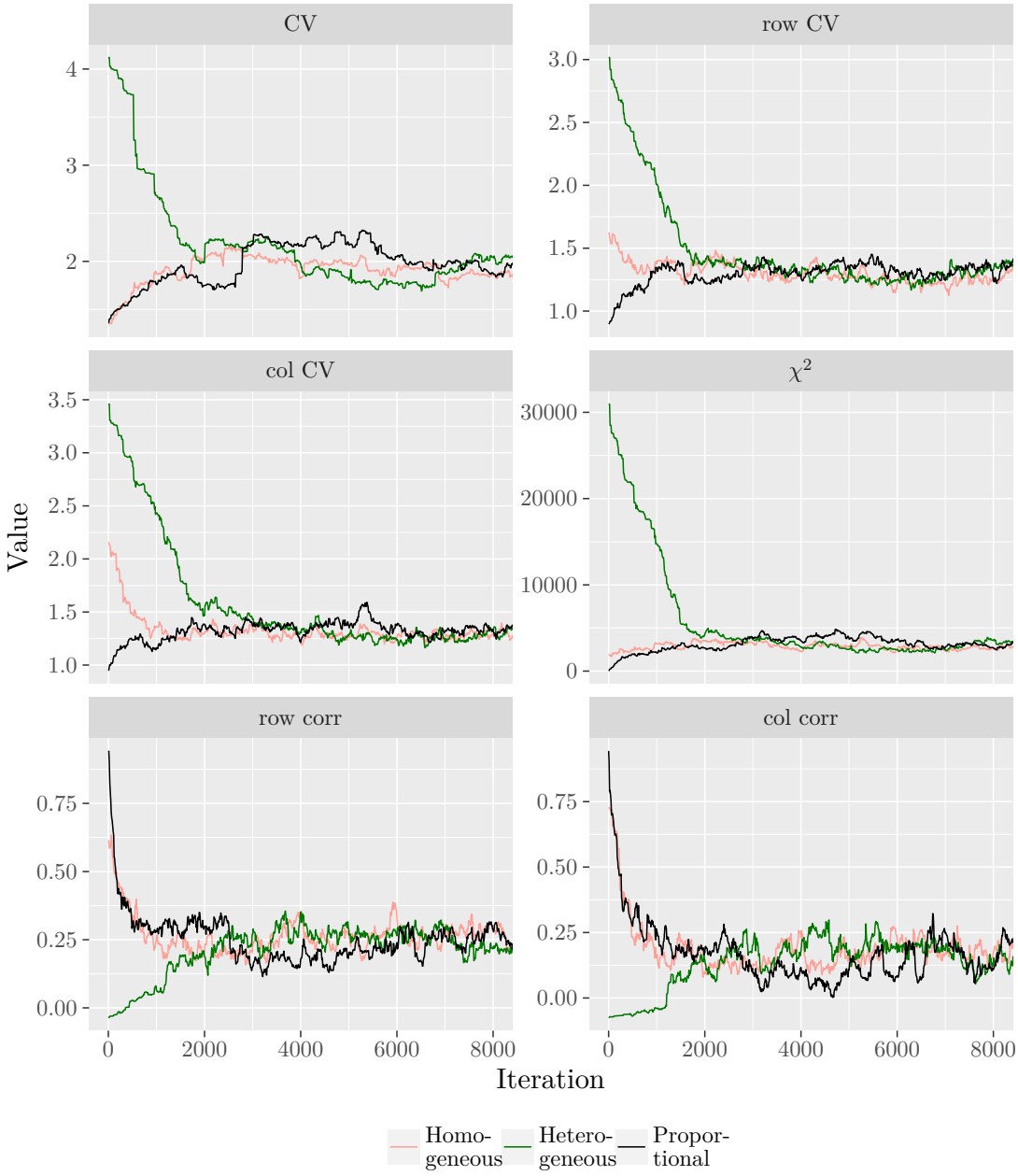


Figure 7.1: Evolution of the measures for a 20×10 matrix, with $N = 20 \times n \times m = 4\,000$. Initial row and column sums ($\overline{\mu}$ and $\overline{\nu}$) are generated with Algorithm 2.3 (Generate Sequences) without constraints (i.e. $\alpha = 0$ and $\beta = N$). Initial matrices are generated with Algorithms 7.1 (Homogeneous Matrices) to 7.3 (Proportional Matrices) without constraints.

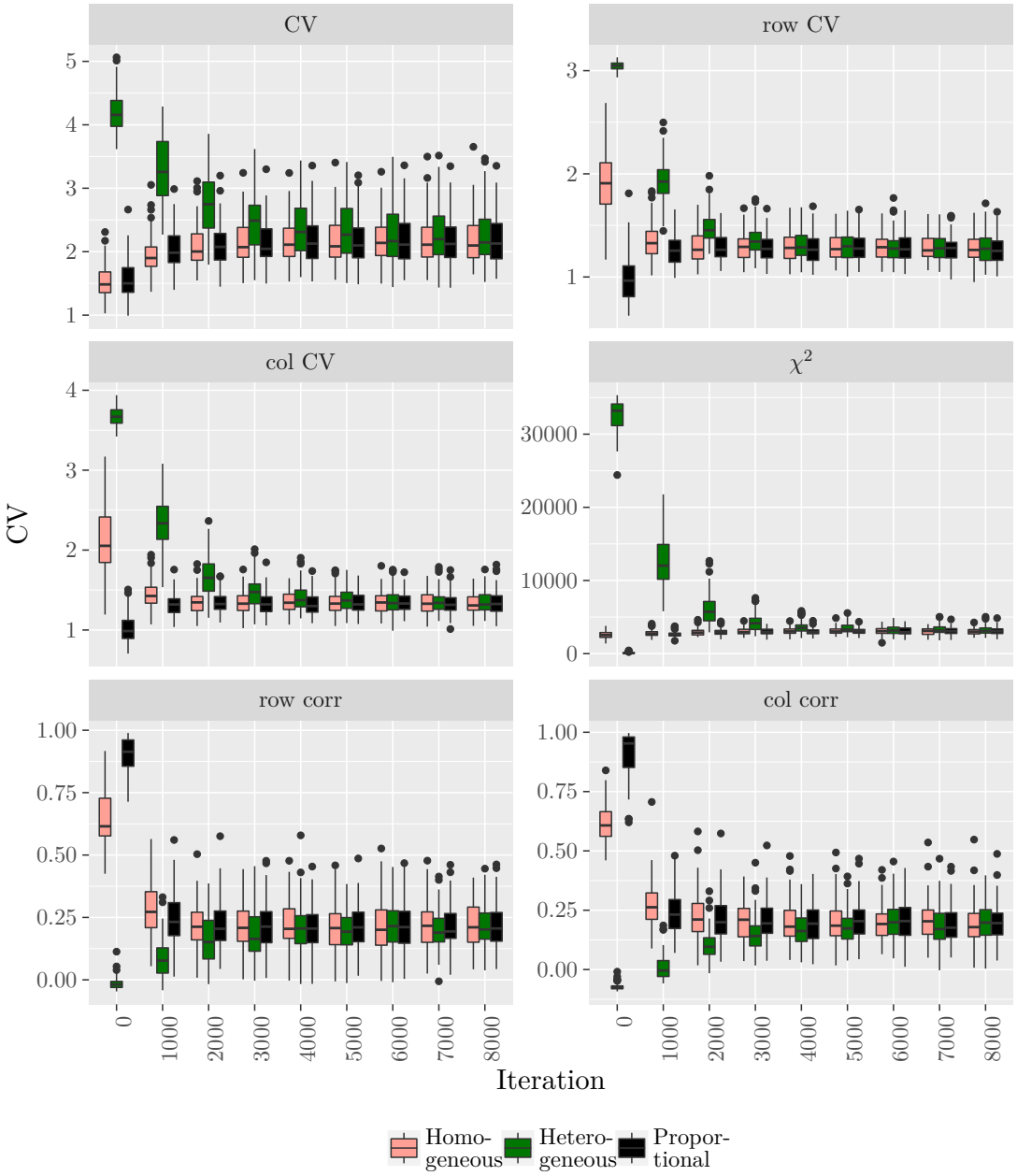


Figure 7.2: Evolution of the measures for matrices with the same characteristics as in Figure 7.1. Each boxplot corresponds to 100 matrices, each based on distinct row and column sums. When a row or column sum is zero, all undefined measures are discarded.

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

7.5 Analysis of Constraints Effect on Cost Matrix Properties

This section studies the effect of the constraints on the matrix properties. The 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 7.4 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 7.3 (Proportional Matrices)).

We want to estimate how the constraints on the $\overline{\mu}$ and $\overline{\nu}$ random generation influence the matrix properties. Figure 7.3 reports the results. Each row is dedicated to a property from the CV to the column correlation that are presented in Section 6.2.2, 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 $\lambda \in \{0, 0.2, \ldots, 1\}$: 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 2.10 in Section 2.4.1: the value decreases from about 20 to 0 as the constraint increases.

The heterogeneity of a cost matrix can be defined in two ways [CP17]: 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 7.4 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) 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. $\lceil \frac{N}{\lambda_c m} \rceil$). 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. Note that when $\lambda = 1$, $A_{\min} = B_{\max}$ and Theorem 7.1 does guarantee the convergence of the MCMC. This is however not an issue because there is a single possible cost matrix in each of these cases.

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. $\lambda_c = 1$).

7.6 Conclusion

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.

Finally, experiments show that constraining the matrix generation with a minimum and maximum matrices leads to large correlations. For instance, with $\lambda_r = 0.9$ and $\lambda_c = 1$, we can generate a 20×10 matrix with a high correlation on columns.

In Chapter 8, we analyze the performance of three scheduling algorithms using

the generated constrained cost matrices and we prove that our experiments are consistent with previous studies in the literature.

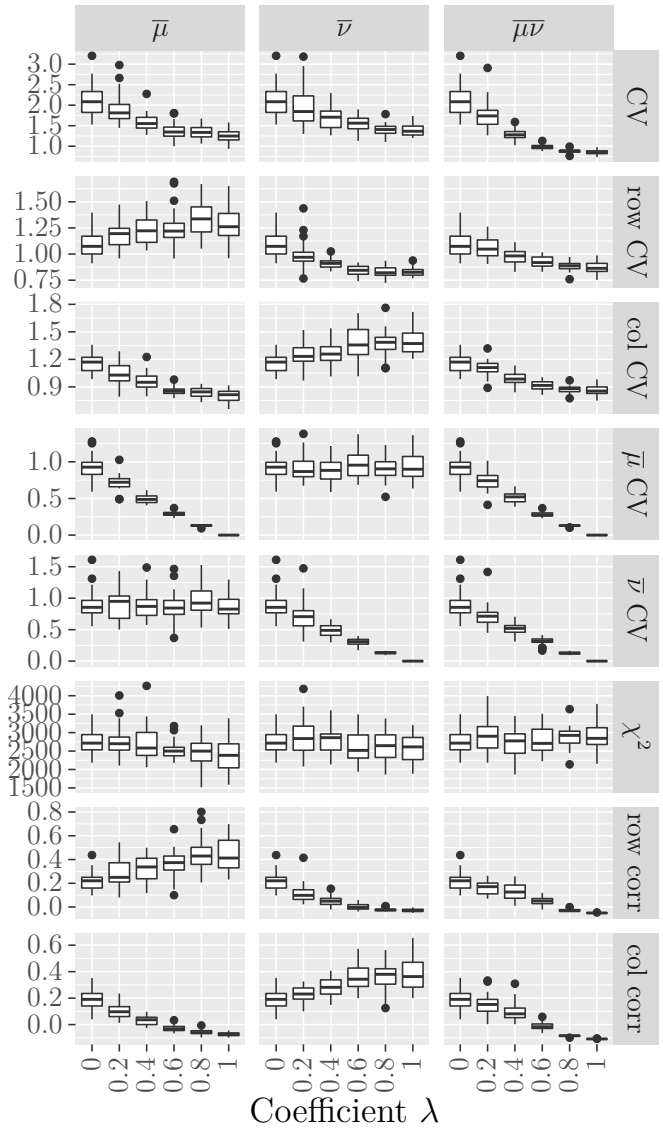


Figure 7.3: Values for the measures presented in Section 6.2.2 and the cost sums ($\overline{\mu}$ and $\overline{\nu}$) CV after 50 000 iterations starting with a proportional 20×10 matrix generated with Algorithm 7.3 (Proportional Matrices) with different constraints on $\overline{\mu}$ and/or $\overline{\nu}$ (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.

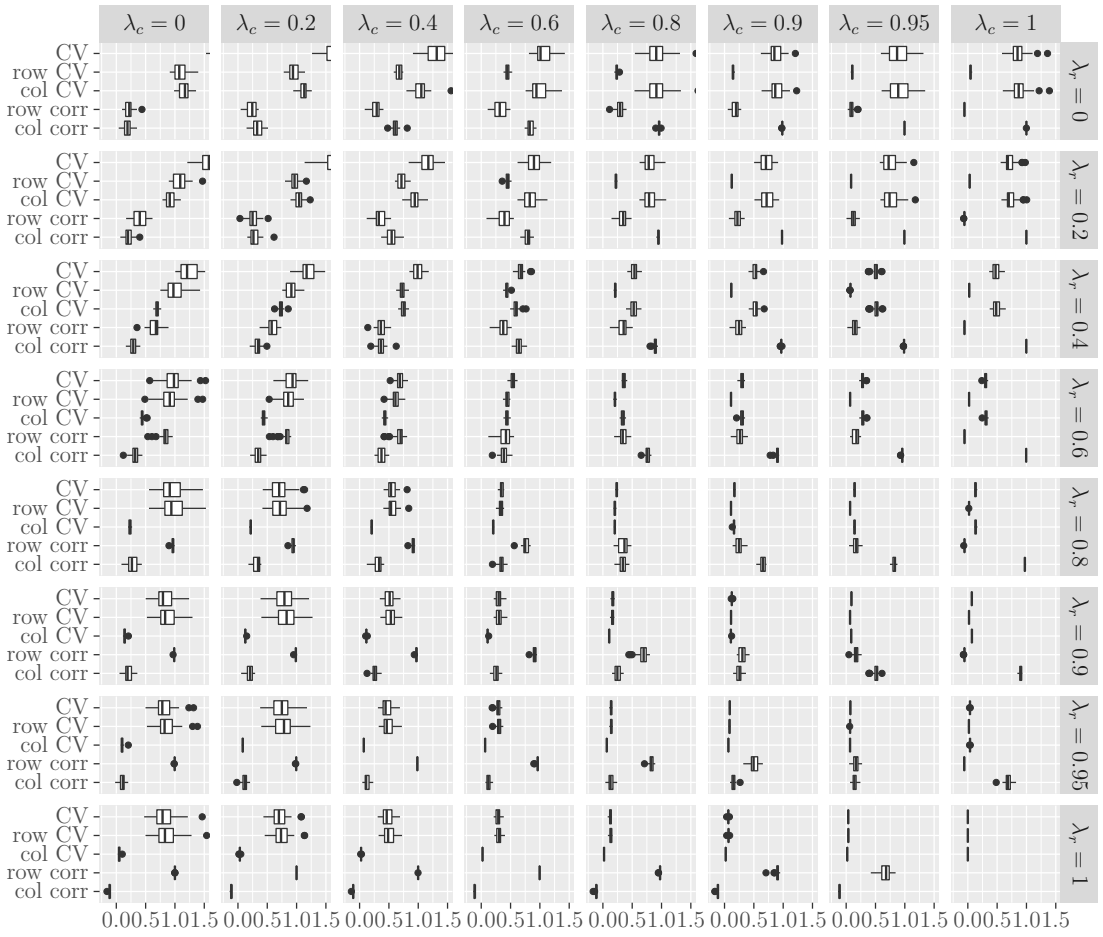


Figure 7.4: Values for the measures presented in Section 6.2.2 and the cost sums ($\overline{\mu}$ and $\overline{\nu}$) CV after 50 000 iterations starting with a proportional 20×10 matrix generated with Algorithm 7.3 (Proportional Matrices) with different constraints on $\overline{\mu}$, $\overline{\nu}$ 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 $0 \leq \lambda_r \leq 1$ (resp. $0 \leq \lambda_c \leq 1$) 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.

Chapter 8

Performance Evaluation of Scheduling Algorithms for Cost Matrices

Contents

In this chapter, Section 8.1 is dedicated to the description of three selected scheduling algorithms: EFT, HLPT, and BalSuff. In Section 8.2, we analyze the effect of the constraints (used for the generation of cost matrices) on each of the selected scheduling algorithm defined in Section 8.1. Section 8.3 is dedicated to the conclusion of this chapter.

8.1 Selected Scheduling Algorithms

Generating random matrices with parameterized constraints allows the assessment of existing scheduling algorithms in different contexts. In our context, we focus on the impact of cost matrix properties on the performance of three heuristics for the problem denoted $R||C_{max}$. Recall that the problem $R||C_{max}$ 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 tasks assigned to it.

The heuristics we consider constitute a diversified selection, in terms of principle and cost, among the numerous heuristics that have been proposed for the problem $R||C_{max}$:

EFT Earliest Finish Time (EFT) [IK77b] is a classic scheduling algorithm, which iteratively assigns each task by selecting the task (among the set of ready tasks) that finishes the earliest on any machine. Thus, EFT has a heterogeneous mechanism. The time complexity of EFT is $O(n^2m)$.

HLPT Heterogeneous Longest Processing Time (HLPT) [CP17] is an extension of LPT [Gra69] and variant of HEFT [THW02b]. HLPT performs as the original LPT when machines are uniform. It differs from EFT by considering first the largest tasks instead of the smallest ones based on their minimum cost on any machine. HLPT starts by sorting tasks in decreasing order of their processing times. Then, it tries to find the best allocation for each task depending on the machine load starting from the longest task, with a minimum completion time in $O(nm + n\log(n))$ steps.

BalSuff Sufferage with machine balancing (BalSuff) is an efficient algorithm [CP17] that balances each task to minimize the makespan. The BalSuff algorithm uses the sufferage matrix in which each value is the difference between the processing time on the current machine and the minimum processing time of the task. The higher the value the more the task will suffer from being mapped on this machine. A null value indicates that the task does not suffer, i.e. the machine processes the task in shortest time. First, BalSuff starts by mapping tasks on their best machine. Second, BalSuff tries to rearrange the tasks in a way such that the makespan is improved and the chosen task is the one that suffers the least from moving. The algorithm stops when there is no more tasks on the most loaded machine that could benefit from moving. Note that if two tasks have the same sufferage value on the most loaded machine the tasks are considered in an arbitrary order (order of the task list of the machine). The algorithm BalSuff has an unknown complexity.

8.2 Analysis of Constraints Effect on Scheduling Algorithms

We selected 25 scenarios that represent different combinations in terms of parameters, heterogeneity and correlation. The constraint on $\overline{\mu}$ and $\overline{\nu}$ 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. $\lceil \frac{N}{\lambda m} \rceil$). Coefficients λ_r and $\lambda_c \in \{0, 0.25, 0.5, 0.75, 1\}$.

We have extreme cases when λ_r (resp. λ_c) is 0 or 1:

- $\lambda_r = \lambda_c = 0$ represents the most heterogeneity and the least correlation.
- $\lambda_r = 0$ and $\lambda_c = 1$ represents a high task and low machine heterogeneity, a low task and high machine correlation.
- $\lambda_r = 1$ and $\lambda_c = 0$ represents a low task and high machine heterogeneity, a high task and low machine correlation.
- $\lambda_r = \lambda_c = 1$ represents identical costs for all tasks on any machine.

Figure 8.1 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 depend on the scenario.

BalSuff outperforms both competitors except in the top right of the figure. Moreover, when $\lambda_r = 0.75$ and $\lambda_c = 1$, BalSuff is even the worst in the median case. 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 [Gra69]. The near-optimality of HLPT for instances with large row and low column heterogeneity is consistent with the literature [CP17]. Finally, EFT performs poorly except when $\lambda_r = 1$. 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 [CHP17]. The case $\lambda_r = \lambda_c = 1$ leads to identical costs for which all heuristics perform the same.

8.3 Conclusion

To see the impact of the properties of the random cost matrices generated with constraints (for the problem $R||C_{max}$), we evaluate the performance of three different scheduling algorithms: EFT, HLPT, and BalSuff.

As we see in Section 8.2, we selected 25 different senarios in terms of correlation and heterogeneity. In some cases, one of the scheduling algorithms outperforms the others, and in other cases its performance changes. The performance of BalSuff and HLPT tends to be the same. When the constraint on columns is very high, HLPT outperforms Balsuff in some cases, and in the rest it is the opposite.

This result is interesting. It shows that the scheduling algorithms, that we tested, have effectively varied relative performance depending on parameters used to constrain the random generation of cost matrices.

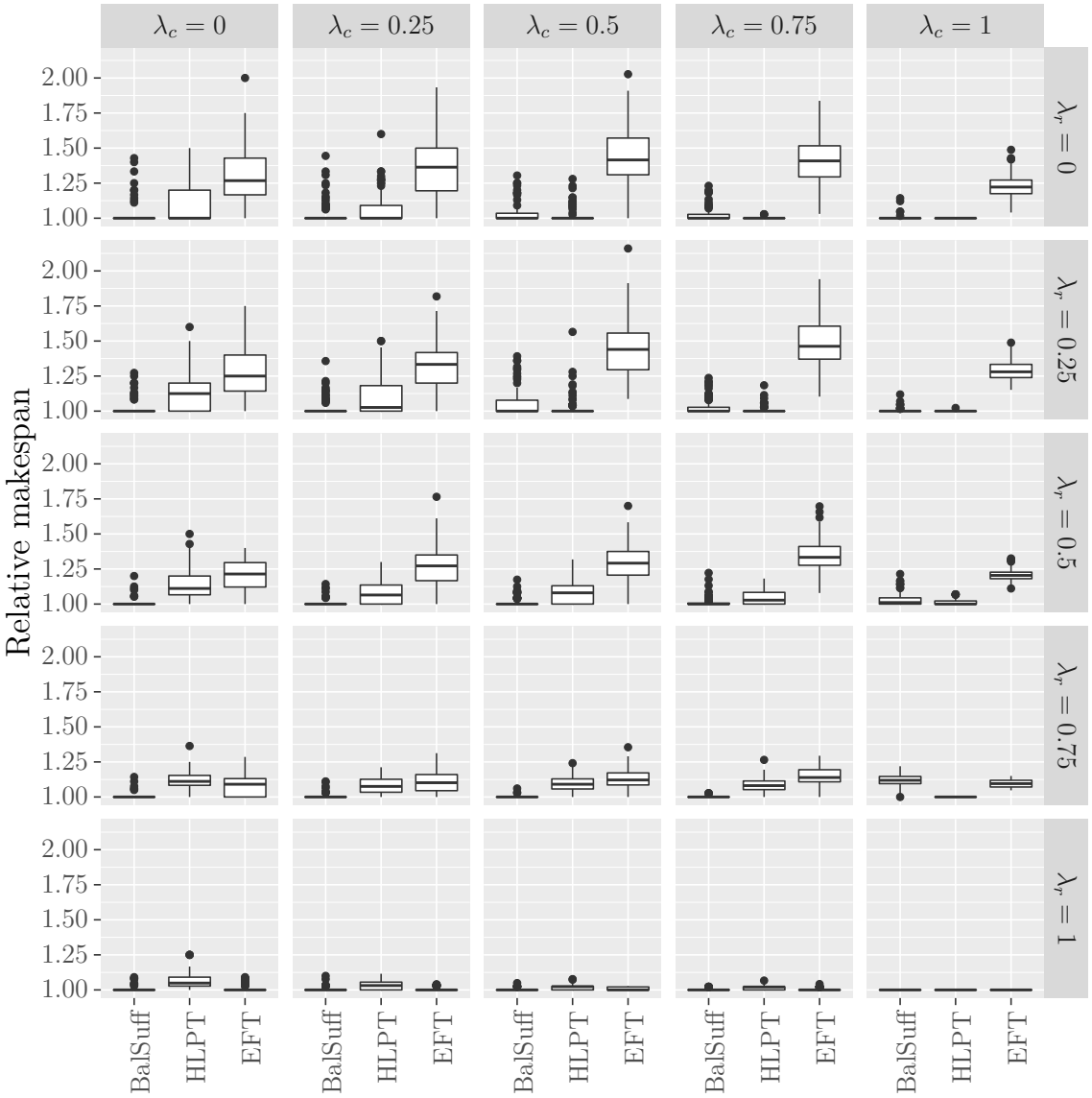


Figure 8.1: Ratios of makespan to the best among BalSuff, HLPT and EFT. The cost sums ($\overline{\mu}$ and $\overline{\nu}$) CV after 50 000 iterations starting with a proportional 20×10 matrix generated with Algorithm 7.3 (Proportional Matrices) with different constraints on $\overline{\mu}$ and $\overline{\nu}$ and with $N = 20 \times n \times m = 4\,000$. The constraint on $\overline{\mu}$ (resp. $\overline{\nu}$) is parameterized by a coefficient $0 \leq \lambda_r \leq 1$ (resp. $0 \leq \lambda_c \leq 1$) 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 100 matrices, each based on distinct row and column sums. When $\lambda_r = \lambda_c = 1$, all costs are identical and the correlations are discarded.

Conclusion of Part III

Regarding the study of cost matrices instances, our work focuses on the generation of cost matrices that 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 method to randomly generate cost matrices from a uniform distribution. We suffle, at each iteration, some costs in the matrix such that the sums of the costs on each row and column remain unchanged. We formally prove that the Markov Chain is ergodic and symmetric, thus 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, we study the impact of our generator on scheduling heuristics (EFT, HLPT, and BalSuff) with heterogeneous costs and processors and we highlight the influence of the parameters of the generator on the performance of these heuristics.

Part IV

General Conclusions and Perspectives

Chapter 9

General Conclusion and Perspectives

Contents

9.1 General Conclusion

As a reminder, this thesis is focused on essentially four questions related to the following ideas: 1) identify the properties of instances (DAGs and cost matrices), 2) analyze the behavior of existing random generators of instances according to the identified properties, 3) ensure a uniform distribution among instances that have given properties, 4) evaluate the performance of scheduling heuristics regarding random generators of both types of instances.

Regarding the first question, this thesis provides answers more specifically for DAGs than for cost matrices. For DAGs, we identify 34 properties and focus on a selection of 8 such properties. We show that the mass property quantifies how much an instance can be decomposed into smaller ones. Concerning cost matrices, we show that our focus is on two properties: the correlation and heterogeneity. In order to represent these properties in a meaningful way, we quantify them using statistical measures.

Regarding the second question, we analyze four different existing random

generators of DAGs: the Erdős-Rényi, a uniform random generation method, random orders, and a layer-by-layer generator. Based on this analysis, we establish the sub-exponential generic time complexity for decomposable scheduling problems with uniform DAGs. For cost matrices, we see that the methods used in the literature to generate cost matrices are biased. Hence, we mention the need for a generator that guarantees a uniform distribution among the set of cost matrices that have a given task and machine heterogeneity.

Regarding the third question, the thesis provides answers for cost matrices and not for DAGs. We propose a new random generator of cost matrices. Our generator relies on Markov Chain Monte Carlo approach to draw random instances with constraints from a uniform distribution. Moreover, we formally prove that the Markov Chain is ergodic and symmetric. Thus, we ensure a uniform distribution over the set of cost matrices generated randomly.

Regarding the fourth question, we highlight the impact of the four random generators of DAGs on three scheduling heuristics (HEFT, HCPT, and MinMin) for a scheduling problem with dependent tasks and we show the need for better generation methods that control multiple properties while avoiding any generation bias. For cost matrices, we study, the impact of our proposed random generator of cost matrices on three scheduling heuristics (EFT, HLPT, and BalSuff) for a scheduling problem with independent tasks. We highlight that for these scheduling heuristics, the performance varies depending on the parameters used to constrain the random generation.

9.2 Perspectives

In the following, we would propose, as perspectives, further investigations concerning DAGs and cost matrices.

P1: As we see in Table 1.4, this thesis does not include a study concerning the proposition of a new random generator of DAGs that can produce instances according to specific constraints on properties. Therefore, it seems logical as a perspective to propose a new method that can uniformly at random generate DAGs with constraints on some properties to avoid pathological instances that are straightforward to solve.

Regarding the generation, for instance, we could generate uniformly at random DAGs with constraints using MCMC approach as we did in the generation of cost matrices in Chapter 7.

Regarding the constraints, let us discuss an example. If a DAG has a width lower than the number of processors, then the problem is easy to solve. In contrast, when the length of a DAG is low, tasks are almost independent and the DAG is not difficult for the problem. Therefore, it seems important to constrain the uniform generation of DAGs such that the length can be controlled.

P2: As a second perspective, we could extend our study to real applications. Practically, all instances generated in our study are random ones. Thus, it is natural to wonder how realistic these instances are. Note that this perspective needs a lot of time to be applied because we will have to analyze real applications (which is not simple to obtain), then make a performance evaluation.

P3: As a third perspective, we could communicate a reference data set with instances that we have preselected (metadata) and make this data set available on an online open access repository (sustainable website) like *figshare*, so that it can be used by researchers to evaluate their algorithms.

Bibliography

[AAK+13] Deepak Ajwani, Shoukat Ali, Kostas Katrinis, Cheng-Hong Li, Alfred J Park, John P Morrison, and Eugen Schenfeld. Generating synthetic task graphs for simulating stream computing systems. *Journal of Parallel and Distributed Computing*, 73(10):1362–1374, 2013.

[AAP+16] Mishra Ashish, Sharma Aditya, Verma Pranet, Abhijit R Asati, and Raju Kota Solomon. A modular approach to random task graph generation. *Indian Journal of Science and Technology*, 9(8), 2016.

[AB14] Hamid Arabnejad and Jorge G Barbosa. List scheduling algorithm for heterogeneous systems by an optimistic cost table. *IEEE Transactions on Parallel and Distributed Systems*, 25(3):682–694, 2014.

[ACD74] Thomas L Adam, K. Mani Chandy, and JR Dickson. A comparison of list schedules for parallel processing systems. *Communications of the ACM*, 17(12):685–690, 1974.

[AGU72] Alfred V. Aho, Michael R Garey, and Jeffrey D. Ullman. The transitive reduction of a directed graph. *SIAM Journal on Computing*, 1(2):131–137, 1972.

[AJ97] Robert K Armstrong Jr. Investigation of effect of different run-time distributions on smartnet performance. Technical report, NAVAL POSTGRADUATE SCHOOL MONTEREY CA, 1997.

[AK98] Ishfaq Ahmad and Yu-Kwong Kwok. On exploiting task duplication in parallel program scheduling. *IEEE Transactions on Parallel & Distributed Systems*, 9:872–892, 1998.

[AKN05] Mona Aggarwal, Robert D Kent, and Alioune Ngom. Genetic algorithm based scheduler for computational grids. In *High Performance Computing Systems and Applications, 2005. HPCS 2005. 19th International Symposium on*, pages 209–215. IEEE, 2005.

[AM11] DI George Amalarethinam and GJ Joyce Mary. DAGEN-A Tool To Generate Arbitrary Directed Acyclic Graphs Used For Multiprocessor Scheduling. *International Journal of Research and Reviews in Computer Science*, 2(3):782, 2011.

[ASM+00] Shoukat Ali, Howard Jay Siegel, Muthucumaru Maheswaran, Debra Hensgen, and Sahra Ali. Representing task and machine heterogeneities for heterogeneous computing systems. *Tamkang J. Sci. Engineer.*, 3(3):195–208, 2000.

[ASMH00] Shoukat Ali, Howard Jay Siegel, Muthucumaru Maheswaran, and Debra Hensgen. Task execution time modeling for heterogeneous computing systems. In *Heterogeneous Computing Workshop (HCW)*, pages 185–199. IEEE, 2000.

[AVÁM92] Virgílio AF Almeida, IMM Vasconcelos, Jose Nagib Cotrim Árabe, and Daniel A Menascé. Using random task graphs to investigate the potential benefits of heterogeneity in parallel systems. In *Proceedings of the 1992 ACM/IEEE conference on Supercomputing*, pages 683–691. IEEE Computer Society Press, 1992.

[BCHC09] Jacob Benesty, Jingdong Chen, Yiteng Huang, and Israel Cohen. Pearson correlation coefficient. In *Noise reduction in speech processing*, pages 1–4. Springer, 2009.

[BLDMK14] Bruno Bodin, Youen Lesparre, Jean-Marc Delosme, and Alix Munier-Kordon. Fast and efficient dataflow graph generation. In *Proceedings of the 17th International Workshop on Software and Compilers for Embedded Systems*, pages 40–49. ACM, 2014.

[Bol01] Béla Bollobás. *Random Graphs*. Cambridge University Press, 2001.

[CDB+16] Pedro Campos, Nizar Dahir, Colin Bonney, Martin Trefzer, Andy Tyrrell, and Gianluca Tempesti. Xl-stage: A cross-layer scalable tool for graph generation, evaluation and implementation. In *Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS), 2016 International Conference on*, pages 354–359. IEEE, 2016.

[CDG+06] Mary Cryan, Martin Dyer, Leslie Ann Goldberg, Mark Jerrum, and Russell Martin. Rapidly mixing markov chains for sampling contingency tables with a constant number of rows. *SIAM Journal on Comp.*, 36:247–278, 2006.

[CDHL05] Yuguo Chen, Persi Diaconis, Susan P Holmes, and Jun S Liu. Sequential monte carlo methods for statistical analysis of tables. *Journal of the American Statistical Association*, 100(469):109–120, 2005.

[CESH18] Louis-Claude Canon, Mohamad El Sayah, and Pierre-Cyrille Héam. A Markov Chain Monte Carlo Approach to Cost Matrix Generation for Scheduling Performance Evaluation. In *International Conference on High Performance Computing & Simulation (HPCS)*, 2018.

[CHP17] Louis-Claude Canon, Pierre-Cyrille Héam, and Laurent Philippe. Controlling the correlation of cost matrices to assess scheduling algorithm performance on heterogeneous platforms. *Concurrency and Computation: Practice and Experience*, 29(15), 2017.

[CLRS09] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

[CMP+10] Daniel Cordeiro, Grégory Mounié, Swann Perarnau, Denis Trystram, Jean-Marc Vincent, and Frédéric Wagner. Random graph generation for scheduling simulations. In *Proceedings of the 3rd international ICST conference on simulation tools and techniques*, page 60. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2010.

[CMRT88] Michel Cosnard, Mounir Marrakchi, Yves Robert, and Denis Trystram. Parallel gaussian elimination on an mimd computer. *Parallel Computing*, 6(3):275–296, 1988.

[CMSV18] Louis-Claude Canon, Loris Marchal, Bertrand Simon, and Frédéric Vivien. Online scheduling of task graphs on hybrid platforms. In *European Conference on Parallel Processing*, pages 192–204. Springer, 2018.

[CP17] Louis-Claude Canon and Laurent Philippe. On the heterogeneity bias of cost matrices for assessing scheduling algorithms. *IEEE Transactions on Parallel and Distributed Systems*, 28(6):1675–1688, 2017.

[CSH19] Louis-Claude Canon, Mohamad El Sayah, and Pierre-Cyrille Héam. A comparison of random task graph generation methods for scheduling problems. *arXiv preprint arXiv:1902.05808*, 2019.

[DBGL+97] Giuseppe Di Battista, Ashim Garg, Giuseppe Liotta, Roberto Tamassia, Emanuele Tassinari, and Francesco Vargiu. An experimental

comparison of four graph drawing algorithms. *Computational Geometry*, 7(5-6):303–325, 1997.

[DC95] Persi Diaconis and L. Salo Coste. Random walk on contingency tables with mixed row and column sums. Technical report, Harvard University, Department of Mathematics, 1995.

[DFLS04] Philippe Duchon, Philippe Flajolet, Guy Louchard, and Gilles Schaeffer. Boltzmann samplers for the random generation of combinatorial structures. *Combinatorics, Probability & Computing*, 13(4-5):577–625, 2004.

[DG00a] Martin Dyer and Catherine Greenhill. Polynomial-time counting and sampling of two-rowed contingency tables. *Theoretical Computer Science*, 246(1-2):265–278, 2000.

[DG00b] Martin E. Dyer and Catherine S. Greenhill. Polynomial-time counting and sampling of two-rowed contingency tables. *Theor. Comput. Sci.*, 246(1-2):265–278, 2000.

[DNSC09] Pierre-Francois Dutot, Tchimou N'takpé, Frederic Suter, and Henri Casanova. Scheduling parallel task graphs on (almost) homogeneous multicluster platforms. *IEEE Transactions on Parallel and Distributed Systems*, 20(7):940–952, 2009.

[DRW98] Robert P Dick, David L Rhodes, and Wayne Wolf. TGFF: task graphs for free. In *Proceedings of the 6th international workshop on Hardware/software codesign*, pages 97–101. IEEE Computer Society, 1998.

[DS+98] Persi Diaconis, Bernd Sturmfels, et al. Algebraic algorithms for sampling from conditional distributions. *The Annals of statistics*, 26(1):363–397, 1998.

[DŠTR12] Tatjana Davidović, Milica Šelmić, Dušan Teodorović, and Dušan Ramljak. Bee colony optimization for scheduling independent tasks to identical processors. *Journal of heuristics*, 18(4):549–569, 2012.

[DZ99] Alain Denise and Paul Zimmermann. Uniform random generation of decomposable structures using floating-point arithmetic. *Theor. Comput. Sci.*, 218(2):233–248, 1999.

[EM12] Daniel Eaton and Kevin Murphy. Bayesian structure learning using dynamic programming and mcmc. *arXiv preprint arXiv:1206.5247*, 2012.

[ER59] P Erdős and Alfréd Rényi. On random graphs I. *Publ. Math. Debrecen*, 6:290–297, 1959.

[Fag76] Ronald Fagin. Probabilities on finite models. *J. Symb. Log.*, 41(1):50–58, 1976.

[FGA+98] Richard F Freund, Michael Gherrity, Stephen Ambrosius, Mark Campbell, Mike Halderman, Debra Hensgen, Elaine Keith, Taylor Kidd, Matt Kussow, John D Lima, Francesca Mirabile, Lantz Moore, Brad Rust, and Howard Jay Siegel. Scheduling resources in multi-user, heterogeneous, computing environments with SmartNet. In *Heterogeneous Computing Workshop (HCW)*, pages 184–199. IEEE, 1998.

[Fis12] George S Fishman. Counting contingency tables via multistage markov chain monte carlo. *Journal of Computational and Graphical Statistics*, 21(3):713–738, 2012.

[FJF16] Lester Randolph Ford Jr and Delbert Ray Fulkerson. *Flows in networks*. Princeton university press, 2016. First edition in 1962.

[FS09] Philippe Flajolet and Robert Sedgewick. *Analytic combinatorics*. cambridge University press, 2009.

[GCJ17] Indrajeet Gupta, Anubhav Choudhary, and Prasanta K Jana. Generation and proliferation of random directed acyclic graphs for workflow scheduling problem. In *Proceedings of the 7th International Conference on Computer and Communication Technology*, pages 123–127. ACM, 2017.

[GJ78] M.R. Garey and D.S. Johnson. Strong NP-completeness results: motivation, examples, and implications. *J. Assoc. Comput. Mach.*, 25(3):499–508, 1978.

[GLLK79] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[GPH83] John Geweke and Susan Porter-Hudak. The estimation and application of long memory time series models. *Journal of time series analysis*, 4(4):221–238, 1983.

[GR+92] Andrew Gelman, Donald B Rubin, et al. Inference from iterative simulation using multiple sequences. *Statistical science*, 7(4):457–472, 1992.

[GR14] Ankur Gupta and James B Rawlings. Comparison of parameter estimation methods in stochastic chemical kinetic models: examples in systems biology. *AIChE Journal*, 60(4):1253–1268, 2014.

[Gra69] R. L. Graham. Bounds on Multiprocessing Timing Anomalies. *Journal of Applied Mathematics*, 17(2):416–429, 1969.

[HJ03] Tarek Hagras and Jan Janecek. A simple scheduling heuristic for heterogeneous computing environments. In *International Symposium on Parallel and Distributed Computing*, page 104. IEEE, 2003.

[IC02] Jaime Shinsuke Ide and Fábio Gagliardi Cozman. Random generation of bayesian networks. In *Advances in Artificial Intelligence, 16th Brazilian Symposium on Artificial Intelligence, SBIA 2002, Porto de Galinhas/Recife, Brazil, November 11-14, 2002, Proceedings*, Lecture Notes in Computer Science, pages 366–375, 2002.

[IK77a] Oscar H. Ibarra and Chul E. Kim. Heuristic Algorithms for Scheduling Independent Tasks on Nonidentical Processors. *Journal of the ACM*, 24(2):280–289, April 1977.

[IK77b] Oscar H Ibarra and Chul E Kim. Heuristic algorithms for scheduling independent tasks on nonidentical processors. *Journal of the ACM (JACM)*, 24(2):280–289, 1977.

[IT07] E Ilavarasan and P Thambidurai. Low complexity performance effective task scheduling algorithm for heterogeneous computing environments. *Journal of Computer sciences*, 3(2):94–103, 2007.

[JCD+13] Gideon Juve, Ann Chervenak, Ewa Deelman, Shishir Bharathi, Gaurang Mehta, and Karan Vahi. Characterizing and profiling scientific workflows. *Future Generation Computer Systems*, 29(3):682–692, 2013.

[KA99] Yu-Kwong Kwok and Ishfaq Ahmad. Benchmarking and comparison of the task graph scheduling algorithms. *Journal of Parallel and Distributed Computing*, 59(3):381–422, 1999.

[KA00] Yu-Kwong Kwok and Ishfaq Ahmad. Link contention-constrained scheduling and mapping of tasks and messages to a network of heterogeneous processors. *Cluster Computing*, 3(2):113–124, 2000.

[KM15] Jack Kuipers and Giusi Moffa. Uniform random generation of large acyclic digraphs. *Statistics and Computing*, 25(2):227–242, 2015.

[KN09] Brian Karrer and Mark EJ Newman. Random graph models for directed acyclic networks. *Physical Review E*, 80(4):046110, 2009.

[KSC78] Valentin Fedorovich Kolchin, Boris Aleksandrovich Sevastyanov, and Vladimir Pavlovich Chistyakov. *Random allocations*. Winston, 1978.

[KSD95] Rainer Kolisch, Arno Sprecher, and Andreas Drexl. Characterization and generation of a general class of resource-constrained project scheduling problems. *Management science*, 41(10):1693–1703, 1995.

[LALG13] Jing Li, Kunal Agrawal, Chenyang Lu, and Christopher Gill. Outstanding paper award: Analysis of global edf for parallel tasks. In *Real-Time Systems (ECRTS), 2013 25th Euromicro Conference on*, pages 3–13. IEEE, 2013.

[Leu04] Joseph YT Leung. *Handbook of scheduling: algorithms, models, and performance analysis*. CRC Press, 2004.

[Lis75] Valery Liskovets. On the number of maximal vertices of a random acyclic digraph. *Theory Probab. Appl.*, 20(2):401–409, 1975.

[LKK83] Robert Earl Lord, Janusz S Kowalik, and Swarn P Kumar. Solving linear algebraic equations on an mimd computer. *Journal of the ACM (JACM)*, 30(1):103–117, 1983.

[LPW06] David A. Levin, Yuval Peres, and Elizabeth L. Wilmer. *Markov chains and mixing times*. American Mathematical Society, 2006.

[Man07] Willem Mantel. Problem 28. *Wiskundige Opgaven*, 10(60-61):320, 1907.

[Mar18] Apolinar Velarde Martinez. Synthetic loads analysis of directed acyclic graphs for scheduling tasks. *International Journal of Advanced Computer Science and Applications*, 9(3):347–354, 2018.

[MDB01] Guy Melançon, I. Dutour, and Mireille Bousquet-Mélou. Random generation of directed acyclic graphs. *Electronic Notes in Discrete Mathematics*, 10:202–207, 2001.

[MKI+03] Ron Milo, Nadav Kashtan, Shalev Itzkovitz, Mark EJ Newman, and Uri Alon. On the uniform generation of random graphs with prescribed degree sequences. *arXiv preprint cond-mat/0312028*, 2003.

[MP04] Guy Melançon and Fabrice Philippe. Generating connected acyclic digraphs uniformly at random. *Inf. Process. Lett.*, 90(4):209–213, 2004.

[NW08] Johanna Nichols and Tandy Warnow. Tutorial on computational linguistic phylogeny. *Language and Linguistics Compass*, 2(5):760–820, 2008.

[OVRO+18] Julian Oppermann, Sebastian Vollbrecht, Melanie Reuter-Oppermann, Oliver Sinnen, and Andreas Koch. GeMS: a generator for modulo scheduling problems: work in progress. In *Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, page 7. IEEE Press, 2018.

[Pea04] K. Pearson. On the theory of contengency and its relation to association and normal correlation. *Drapers' Company Reserach Memoirs*, 1904.

[Plo07] Anatoly D Plotnikov. Experimental algorithm for the maximum independent set problem. *arXiv preprint arXiv:0706.3565*, 2007.

[Rob73] R. W. Robinson. Counting labeled acyclic digraphs. In F. Harray, editor, *New Directions in the Theory of Graphs*, pages 239–273, New York, 1973. Academic Press.

[SAHR12] Rishad A Shafik, Bashir M Al-Hashimi, and Jeff S Reeve. System-level design optimization of reliable and low power multiprocessor system-on-chip. *Microelectronics Reliability*, 52(8):1735–1748, 2012.

[SGB06] Sander Stuijk, Marc Geilen, and Twan Basten. SDF^3: SDF for Free. In *Application of Concurrency to System Design, 2006. ACSD 2006. Sixth International Conference on*, pages 276–278. IEEE, 2006.

[SMD11] Boonyarith Saovapakhiran, George Michailidis, and Michael Devetsikiotis. Aggregated-dag scheduling for job flow maximization in heterogeneous cloud computing. In *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE*, pages 1–6. IEEE, 2011.

[Sul07] Danny Sullivan. What is google pagerank? a guide for searchers & webmasters. *SearchEngineLand*, 26:070426–011828, 2007.

[SZ04] Rizos Sakellariou and Henan Zhao. A hybrid heuristic for dag scheduling on heterogeneous systems. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, page 111. IEEE, 2004.

[THW02a] Haluk Topcuoglu, Salim Hariri, and Min-you Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE transactions on parallel and distributed systems*, 13(3):260–274, 2002.

[THW02b] Haluk Topcuoglu, Salim Hariri, and Min-you Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans. on Parallel and Dist. Systems*, 13(3):260–274, 2002.

[TK02] Takao Tobita and Hironori Kasahara. A standard task graph set for fair evaluation of multiprocessor scheduling algorithms. *Journal of Scheduling*, 5(5):379–394, 2002.

[Ull75] J.D. Ullman. NP-complete scheduling problems. *J. Comput. System Sci.*, 10:384–393, 1975.

[WG90] M-Y Wu and Daniel D Gajski. Hypertool: A programming aid for message-passing systems. *IEEE transactions on parallel and distributed systems*, 1(3):330–343, 1990.

[Win85] Peter Winkler. Random orders. *Order*, 1(4):317–331, 1985.

[YG94] Tao Yang and Apostolos Gerasoulis. DSC: Scheduling parallel tasks on an unbounded number of processors. *IEEE Transactions on Parallel and Distributed Systems*, 5(9):951–967, 1994.