



**Georges Miranda Spyrides**

**Branch-cut-and-price approach for Process  
Discovery**

**Dissertação de Mestrado**

Dissertation presented to the Programa de Pós-graduação em  
Informática of PUC-Rio in partial fulfillment of the requirements  
for the degree of Mestre em Informática.

Advisor: Prof. Marcus Vinicius Soledade Poggi de Aragão

Rio de Janeiro  
March 2019



**Georges Miranda Spyrides**

**Branch-cut-and-price approach for Process  
Discovery**

Dissertation presented to the Programa de Pós-graduação em  
Informática of PUC-Rio in partial fulfillment of the requirements  
for the degree of Mestre em Informática. Approved by the  
undersigned Examination Committee.

**Prof. Marcus Vinicius Soledade Poggi de Aragão**

Advisor

Departamento de Informática – PUC-Rio

**Prof. Hélio Côrtes Vieira Lopes**

Departamento de Informática – PUC-Rio

**Prof. Silvio Hamacher**

Departamento de Engenharia Industrial – PUC-Rio

Rio de Janeiro, March 28<sup>th</sup>, 2019



All rights reserved.

### **Georges Miranda Spyrides**

Bachelor's in Industrial Engineering (2013) at the Federal University of Rio de Janeiro (UFRJ). Worked for EloGroup as business consultant from 2013 to 2016. Since 2017 works at GALGOS, former ATD-Lab, at PUC-Rio.

#### Bibliographic data

Spyrides, Georges Miranda

Branch-cut-and-price approach for Process Discovery / Georges Miranda Spyrides; advisor: Marcus Vinicius Soledade Poggi de Aragão. – Rio de Janeiro: PUC-Rio, Departamento de Informática, 2019.

v., 130 f: il. color. ; 30 cm

Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática.

Inclui bibliografia

1. Descoberta de processos. 2. Mineração de processos. 3. Programação Inteira. 4. Geração de coluna. I. Aragão, Marcus Vinicius Soledade Poggi. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

## Acknowledgments

This work would not be possible without the help and support by many special persons.

To my old friends, thanks for the patience in my absence and in my presence, when I was absorbed by all the abstract ideas present in this work.

To my new friends, thank you for turning PUC into a second home. The extended group from Galgos provided an excellent environment for discussing new subjects while sharing a particular kind of humor. In special, I want to thank Rafael, Guilherme, and Beatriz for helping me out in different parts of the implementation of this work.

To all the professors in PUC's Departamento de Informatica, thank you for presenting an extensive collection of new exciting ideas. To Poggi, an advisor that always made possible crossing my research roadblocks through outrageously implausible paths, thanks for the opportunity of working together.

To my family, thanks for all the affection and for providing the means during my formation and especially during this work.

Finally, this work would not be possible without the unconditional support from my beloved Helen. Thank you, dear.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) Finance Code 001 and in part by Fundação de Amparo à Pesquisa do Estado do Rio de Janeiro (FAPERJ).

## Abstract

Spyrides, Georges Miranda; Aragão, Marcus Vinicius Soledade Poggi (Advisor). **Branch-cut-and-price approach for Process Discovery**. Rio de Janeiro, 2019. 130p. Dissertação de mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Process Discovery amounts to determine a process model from an event log of a business process. Many process discovery algorithms try to synthesize a Petri net representing the log by finding places and arcs that relate the event classes. Bergenthum et al. (2007) and van der Werf et al. (2008) propose formulations for this problem discover one place at a time, in which each basic solution of the set of inequalities represents a candidate place. We propose a global integer programming formulation that, given a log, determines all places and arcs defining a Petri net. This model simplifies the selection of places but has an efficiency problem due to a large number of integer variables used. We also propose a decomposition method for the global ILP model to treat each place and their associated constraints as a separate sub-problem. We can run the algorithm on large synthetic instances, which is unprecedented for this kind of process miner.

## Keywords

Process Discovery; Process Mining; Integer Programming; Column generation.

## Resumo

Spyrides, Georges Miranda; Aragão, Marcus Vinicius Soledade Poggi. **Uma abordagem para Mineração de Processos usando geração de colunas e cortes**. Rio de Janeiro, 2019. 130p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Descoberta de Processo significa determinar um modelo de processo a partir de um registro histórico de eventos de um processo de negócios. Muitos algoritmos de descoberta de processos tentam sintetizar uma rede de Petri que representa o registro localizando locais e arcos que relacionam as classes de eventos. Bergenthum et al (2007) e van der Werf et al. (2008) propõem formulações para este problema descobrir um place de cada vez, em que cada solução básica do conjunto de desigualdades representa um lugar candidato. Propomos uma formulação global de programação inteira que, dado um registro histórico, determina todos os places e arcos que definem uma rede de Petri de uma só vez. Este modelo é uma alternativa a seleção de locais, mas tem um problema de eficiência devido à grande quantidade de variáveis inteiras usadas. Também propomos um método de decomposição para o modelo ILP global para tratar cada place e suas restrições associadas como um subproblema separado. Conseguimos então executar o algoritmo em instâncias sintéticas grandes, o que é inédito para esta classe de mineradores de processo.

## Palavras-chave

Descoberta de processos; Mineração de processos; Programação Inteira; Geração de coluna.

## Table of contents

1	Introduction	14
1.1	Motivation and Research questions	18
1.2	Structure of this work	19
2	Basic concepts and definitions	21
2.1	Process mining	21
2.2	Linear Programming Formulation with Parikh vectors	22
2.3	Base formulation for a single place	23
2.4	Challenges of measuring conformance	25
3	Analysis and interpretation of the classic formulation	28
3.1	Unimodularity	28
3.2	Alphabet filtering	32
3.3	Susceptibility to outlier logs	34
4	Global Integer Programming - the compact model	36
4.1	The global model	36
4.2	Global Formulation's auxiliary ideas	39
4.3	Global ILP Experiments	41
4.4	Results	47
5	Branch-Cut-and-Price Global formulation	49
5.1	Place oracle (Column Generation)	51
5.2	Alternative place oracles	55
5.3	Column generation stabilization	57
5.4	Cut generation	59
5.5	Branch and bound	59
5.6	Experimental setup	60
5.7	Results	67
6	Risk-prone generation of places	73
6.1	Cardinality constrained Robust Model	74
6.2	Experiments	77
6.3	Results	78
7	Discussion	81
7.1	Findings and contributions	81
7.2	Limitations and future research	82
	Bibliography	85
A	BCP model computational experiment detailed	89
B	Risk-prone model computational experiment detailed	92

C	Process models generated using the Branch-Cut-and-Price Model	95
D	Process models generated using the Risk-Prone model	115

## List of figures

- Figure 1.1 An illustrated representation of a firing in a given Petri net. In this Petri net it is possible to fire the sequences  $\langle A, B, D \rangle$  and  $\langle A, C, D \rangle$ . Notice that there is an implicit Xor relationship between B and C, because they consume tokens from the same place. 15
- Figure 1.2 Another pictorial representation of firing in a given Petri net. In this Petri net it is possible to fire the sequences  $\langle A, B, C, D, E \rangle$ ,  $\langle A, D, B, C, E \rangle$  and even  $\langle A, B, D, C, E \rangle$ . Observe that there is an implicit AND relationship between B and D because A produces two tokens that allow each of the branches to execute independently. Furthermore, the two places arriving at E represent the wait to finish each of the parallel branches of the process. Parallelism manifests itself as permutations in the set of possible firing sequences. 16
- Figure 2.1 An example of the solution. We represent with the variables  $x_i$  and  $y_j$  the presence or absence of an arc incoming or going to a transition respectively 24
- Figure 2.2 Viable solutions of the classic formulation using a single sequence language  $\langle A, B, C \rangle$ . Observe that there are many places that satisfy the constraints from the model. Also observe that imposing a minimal number of arcs yield the most simple place candidates. 24
- Figure 2.3 An example of the star model from (Rozinat2007). Every sequence of every length is possible using the transitions that have incoming and ongoing arcs to the same place. 26
- Figure 2.4 An example of a linear enumeration model from (Rozinat2007) . We can simply build linear models using the language and an exclusive choice at the beginning to achieve maximum fitness and maximum precision. We have also to take into consideration the simplicity of the model as a function of the number of arcs and transitions of the same symbol used. 27
- Figure 2.5 An example of a balanced petri net from (Rozinat2007). It does not achieve maximum fitness or precision, but it is simple. The consequence of balancing is a model in which the connections denote parallelism, decisions and other behaviors useful for a business process analyst clearly. 27

Figure 3.1 Prefixes transform into the left-hand side of the constraint inequalities. If the set is a closure of prefixes, we can guarantee that for each sequence with length above one symbol will have its immediate prefix within the closure. By immediate prefix we mean the sequence with just the last symbol missing of a giving sequence. Thus, if the last symbol of the sequence  $\sigma_1$  is  $a_{i+1}$  and the last symbol of its immediate prefix sequence  $\sigma_2$  is  $a_i$ , we expect the left-hand side of the inequality associated to  $\sigma_1$  be the same as the inequality associated to  $\sigma_2$  added  $+x_{a_i} - y_{a_{i+1}}$ . 30

Figure 3.2 Many heuristics could be used to factor the left-hand side matrix of the formulation. The proof we have given based on immediate prefixes is the alternative example at the bottom of this picture. 32

Figure 4.1 Graphical representation of the layers imagined for the global formulation. Here the original  $x_i^k$  and  $y_j^k$  variables are depicted as a lower level in which we search for a place  $k$ . The upper layer is a direct network between events, connecting directly an event  $i$  to another  $j$  represented by  $w_{ij}$  variables. Then there is the middle layer in which translates  $x_i^k$  and  $y_j^k$  into a direct connection  $a_{ij}$  between transition  $i$  that goes to place  $k$  and transitions  $j$  that arrive from place  $k$ . The problem becomes to cover all  $w_{ij}$  using available  $a_{ij}$ . 38

Figure 4.2 Results table. It compares the outcomes of each of the 14 small synthetic instances to 9 variant setups. It is possible to deduce that the auxiliary ideas in the formulation do not help much and that the intrinsic difficulty lies within the instance itself. 47

Figure 4.3 Execution times of the experiments in seconds. Across variants, the experiment without fixing initial variables took much longer. The degree of freedom in the  $w_{ij}$  network increases severely the execution times. 48

Figure 5.1 Graphical representation of the Global Model non-zeros of the formulation. The main idea was to replicate the classical formulation  $k$  times and to enforce cohesion using general constraints on a direct network linking events to events. A block-structure like this with linking constraints suggests the opportunity to apply the Dantzig Wolfe decomposition method for efficiency gain. 50

Figure 5.2 Branch-cut-and-price algorithm schematics. A recursive procedure controls the branching of different fixations made to the relaxed master. There is a linear solver which verifies the reduced cost to call the pricing subproblem and the minimum cut in the  $w_{ij}$  network graph. 50



- Figure 5.3 An example of partitioning the alphabet to form smaller subproblems. The SCC is the first step in good implementations of the Inductive Miner algorithm. We take the advantage that this first step separates the alphabet into subalphabets partitions that envelop inside difficult characteristics such as parallelism and cycles. Each of these strongly connected components can generate pricing problems using the filtered approach described in 3 56
- Figure 5.4 Example of the reproduction of a single log  $\langle a, b, d, e, g \rangle$  in a process model. During the execution, we maintain counters for C, P, M and R. Notice that the process model is flawed to a point where “b” and “g” are not even transitions in the process. Even with this characteristic we carry on forcibly activating the transitions which do belong to the model. 63
- Figure 5.5 Example of a Petri net simulation tree adapted from Adriansyah’s paper on Measuring precision (Adriansyah2015). The internal nodes in white represent prefixes in the simulation of the Petri net which are present in the original event log. In red are the leaving arcs, which mean prefixes that can be generated by the Petri net but are not found in the event log. The approximation of the precision measure is the unweighted version, which is the simple division of internal nodes by the sum of internal nodes and leaving arcs of the tree. 65
- Figure 5.6 Table showing characteristics of each of the tested instances and the evaluation of the best solution found by our algorithm. We evaluated our algorithm by leaving arcs precision, token replay fitness, integrality gap and execution time of the algorithm. 68
- Figure 5.7 Elapsed time by optimization milestone. Before the algorithm starts branching, it finds a root solution. Then when the branching starts, it finds an integer solution (Petri net) and continues branching until it has tested all possibilities of fixating the root solution. Notice that the first solution the algorithm finds serves as bound for the branching procedure, and then the algorithm will only search for branches and solutions promising Petri nets with better objective function value. 69
- Figure 5.8 A scatter plot showing the relationship between the integrality gap and the fitness measure. The algorithm will find integer solutions with lower and lower objective function values relative to the root objective function. Therefore, we can visually interpret this chart as time flowing to the right as the algorithm progresses narrowing the integrality gap, notice that the right corner means gap equal to zero. The formulation for this problem seems to have a low standard integrality gap, and the objective functions seem disconnected to the fitness measure. 70

- Figure 5.9 A scatter plot showing the relationship between the integrality gap and the precision measure. Notice that as the gap reduces, the objective function reduces and so the number of arcs used to connect the Petri nets. The algorithm may eliminate the production of unnecessary tokens that enable regions not present in the log. 71
- Figure 5.10 A scatter plot showing the relationship between the fitness measure and the precision measure. Once we find a solution, it does not seem to be substituted by an extremely better one. In most of the cases, the Petri nets generated seem to be always close in terms of results to one another. 72
- Figure 6.1 Example of the additions made to the previous sub-problem in the branch-cut-and-price schema. For each prefix constraint belonging to the same variant, we add a binary variable that relaxes the problem whenever its value changes from one to zero. An additional constraint is added to control which of the new relaxation binary variables can assume the value 0. The idea is to allow rare variants to be more likely to be shut down. In this particular example, we only allow for the constraints associated with  $\langle A, D, E \rangle$  to be relaxed. 76
- Figure 6.2 Table showing the execution time and Objective function for the execution of the algorithm using different values for  $\eta$ . Notice that the execution time peaks at an  $\eta = 0.7$ . Also, our objective function decreases 10 units when a model has one less arc relative to the others. We observe that the opportunity to reduce the quantity of arcs is relatively little by decreasing values from  $\eta$  (eta). 77
- Figure 6.3 Fitness and Precision measures of the 12 instances as  $\eta$  decreases. An  $\eta$  (eta) of 100% on the left is analogous to results obtained in section 5. We can observe the progressive relaxation of  $\eta$  as the lines of the chart move to the right. A counter-intuitive result: generating places and arcs allowing the model to ignore a large amount of history can yield better precision while maintaining high fitness. 79

*If you can keep your head when all about you  
Are losing theirs and blaming it on you,  
If you can trust yourself when all men doubt you,  
But make allowance for their doubting too;  
If you can wait and not be tired by waiting,  
Or being lied about, don't deal in lies,  
Or being hated, don't give way to hating,  
And yet don't look too good, nor talk too wise:*

*If you can dream—and not make dreams your master;  
If you can think—and not make thoughts your aim;  
If you can meet with Triumph and Disaster  
And treat those two impostors just the same;  
If you can bear to hear the truth you've spoken  
Twisted by knaves to make a trap for fools,  
Or watch the things you gave your life to, broken,  
And stoop and build 'em up with worn-out tools:*

*If you can make one heap of all your winnings  
And risk it on one turn of pitch-and-toss,  
And lose, and start again at your beginnings  
And never breathe a word about your loss;  
If you can force your heart and nerve and sinew  
To serve your turn long after they are gone,  
And so hold on when there is nothing in you  
Except the Will which says to them: 'Hold on!'*

*If you can talk with crowds and keep your virtue,  
Or walk with Kings—nor lose the common touch,  
If neither foes nor loving friends can hurt you,  
If all men count with you, but none too much;  
If you can fill the unforgiving minute  
With sixty seconds' worth of distance run,  
Yours is the Earth and everything that's in it,  
And—which is more—you'll be a Man, my son!*

**Rudyard Kipling, .**

# 1

## Introduction

A process is a transformation phenomenon that manifests itself through a sequence of activities and events seeking a specific goal. There are also decisions made through this process that affects the flow of events. The understanding of a process as a complex system of flow and decision is the first step to its improvement, and, thus, achieving desired goals more efficiently.

It is usual for organizations to record the execution of its activities in logs of events and activities. Complex service industries such as healthcare, insurance, and banking are heavy-users of enterprise resource systems which log activities performed by workers at all times. These event logs contain traces of the execution of hundreds of formal and emerging processes. This work aims to create an algorithm that extracts process rules and logic flow of events from these logs. The name of this problem is Process Discovery.

The main task executed by Process Discovery algorithms is to reconstruct a representation of a business process from its event log, which can re-enact the event sequences back through simulation. Then, the task of discovery is the reverse of simulating: the user wants to discover the rules from the sequences observed. Noisy data or complex flow structures are difficult to express with simplicity. The presence of outlier sequences and some structures such as loops and parallel streams present in the same process are examples.

The primary challenge of process discovery algorithms is to obtain a process model which is reasonably replayable by machines, readable by humans. These mined models should be representative of all and only the sequences of activities present in the event log. Therefore it is usual to think of process representations as a graph or as a flowchart.

Many representations of business processes are possible. The solution strategy of some process discovery algorithms relies heavily on how they encode the process. Some examples of structures used by these algorithms are process trees, C-nets, BPMN flowcharts, and Petri nets.

Petri nets are particular graphs with two kinds of nodes: places and transitions. The arcs of this graph can go from a place to a transition or from a transition to a place. Each place can carry activation tokens, any number of them. Transitions can fire. When they fire, they consume tokens from each of

the places which have incoming arcs to the transition and produce tokens to every place which have incoming arcs from the transition. For this work, we will consider a subvariant of Petri nets which consume and produce only one token for each arc.

In a simulation, a transition can only fire if the places that go to it have at least one token, that means the transition is enabled. It is possible to carry a simulation of a Petri net by randomly firing any enabled transition and stopping when no transition is longer available.

Notice that after a transition fires, they modify the state of the Petri net: the count of tokens in each arc. Thus, after the firing, the set of enabled transitions also change until it is empty. Therefore a given fixed Petri net has an enumerable deterministic set of possible firing sequences. Furthermore, if the Petri net has cycles, this set is possibly infinite.

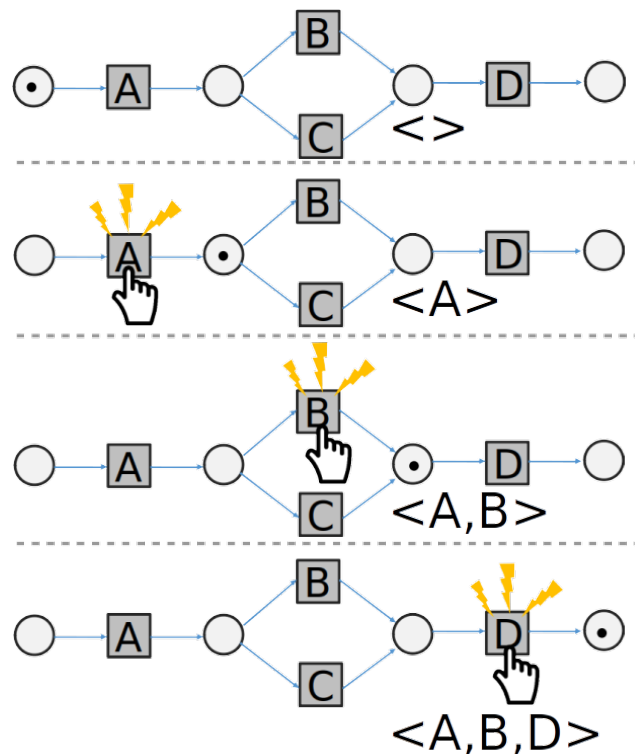


Figure 1.1: An illustrated representation of a firing in a given Petri net. In this Petri net it is possible to fire the sequences  $\langle A, B, D \rangle$  and  $\langle A, C, D \rangle$ . Notice that there is an implicit Xor relationship between B and C, because they consume tokens from the same place.

The Petri net describes behavior based on how places and transitions are connected. Some examples are: places that have arcs going to multiple transitions describe an exclusive decision behavior, transitions that have arcs going to multiple places describe the opening of parallel flow, transitions that

have multiple arcs coming from different places describe a relationship of multiple pre-requisites for firing that specific transition.

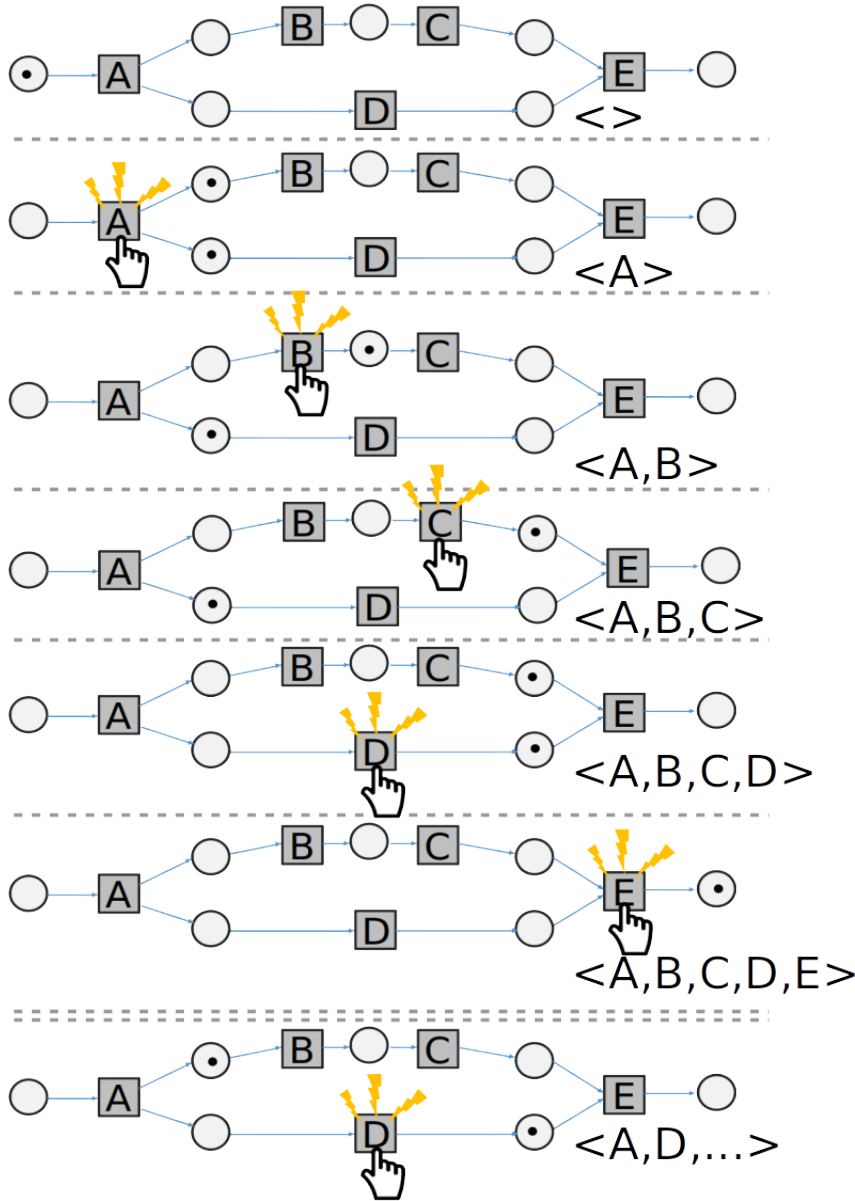


Figure 1.2: Another pictorial representation of firing in a given Petri net. In this Petri net it is possible to fire the sequences  $\langle A, B, C, D, E \rangle$ ,  $\langle A, D, B, C, E \rangle$  and even  $\langle A, B, D, C, E \rangle$ . Observe that there is an implicit AND relationship between B and D because A produces two tokens that allow each of the branches to execute independently. Furthermore, the two places arriving at E represent the wait to finish each of the parallel branches of the process. Parallelism manifests itself as permutations in the set of possible firing sequences.

Formally, we also can see the Petri net as a directed bipartite graph containing a set of transitions and a set of places that connect one to another. For readers interested in deepening their understanding, we refer to Reisig's excellent explanations on (Reisig2013).

Algorithms using Petri nets already have the set of transitions and their firing sequences, then they try to propose places and their respective connections to the given set of transitions in order to synthesize a Petri net. These algorithms usually rely on heuristics built upon the theory of regions to find the set of places and how they connect with the set transitions. In the context of this research, we are interested in obtaining a Petri net from an event log.

One of the landmarks in the field of process mining is the  $\alpha$ -algorithm. This algorithm uses a greedy strategy to propose places and their respective connections iteratively. It begins preprocessing the logs into a matrix composed of the precedence frequency of activities. For instance, in all the log if the activity  $A$  was followed five times by activity  $B$  the element  $f_{AB}$  of the frequency matrix  $F$  is equal to 5.

Then it builds a set of pairs of activities (or edges) which are exclusively sequential, that is, a pair of activities  $(i, j)$  is included if  $f_{ij} > 0$  AND  $f_{ji} = 0$ . From this set, a subroutine searches for subsets of edges (pairs of activities) which form a complete bipartite subgraph.

Bergenthum et al (Bergenthum2007) proposes a formulation using the Parikh vectors over a prefix-closed language associated with the event log. They also discuss how the polyhedron defined by a system of inequalities derived with the theory of regions approach has its vertices (basic solutions) related to a place in the process to be synthesized as a Petri net. Moreover, they discuss procedures to eliminate redundant places synthesized in the Petri net.

The same approach is further developed by van der Werf et al. (VanDerWerf2008) and (VanDerWerf2009). They present the formulation of the ILP Miner and advances the previous work proposing methods to search for a final marking. Thus, the formulation proposed returns a single candidate place for each optimization step. The resulting algorithm constructs a Petri net by repeatedly solving the ILP in search of additional places. Since then, ILP is used as a benchmark method, because it has useful properties. Properties such as its formal guarantee of perfect fitness and dealing efficiently with big event logs, assuming they do not deal with a wide variety of activities (VanZelst2015).

We will show in this article a new way of approaching the problem based on the decomposition of this formulation. We will also show that the formulation is totally unimodular and that after a few row operations we can always arrive at a specific formulation in which we can understand the differences of all algorithms exploiting the theory of regions to find places.

Generally speaking, the ILP answer to the process discovery has still a great practical use. However, even with the stunning improvement of the efficiency of commercial solvers for integer programs, the resolution of ILPs may be prohibitive for some problems structures and huge sizes.

Often, researchers in the area address the scalability problem with a large variety of events not just with the ILP method. Verbeek, in (Verbeek2017) and (Verbeek2014), addresses the scaling problem. In both articles, the authors explore the efficiency problem by subdividing logs into chunks. Then, the miner finds solutions significant less time.

The same approach is used in different settings in (Carmona2010) and (Ekanayake2013). In (Carmona2010), Carmona uses the identification of groups through PCA and firing sequence causalities then using intra- and inter-group constraint formulation. In (Ekanayake2013), Ekanayake uses hierarchical trace clustering to identify subprocesses. Van der Aalst in (VanZelst2015) makes another attempt at simplifying the model. It tries to solve differently, proposing a hybrid of formulations using a more robust formulation only when cycles are present. Therefore, they reduce the number of integer variables needed in previous ILP formulations.

We advance in a way to treat the cohesion between places as a mathematical programming model. This advance allows future research to fuse algorithmic ideas used in other methods, by providing the flexibility of modeling these ideas as mathematical programs. Our motivation is to give the global formulation the needed efficiency to justify its use in practice. Therefore, we propose a decomposition method that exploits the block-structured characteristic of the problem.

## 1.1

### Motivation and Research questions

The central questions that guided this work was: “How to derive the whole Petri net from a single integer formulation?”. Previous attempts used a formulation that returned just a single place and relied on other strategies to choose among these places to form a Petri net. We were able to devise a global formulation, but it was not time-efficient for medium- and large-sized problem instances. Compared to other process discovery algorithms, this approach is very time-consuming.

Also, algorithms based on places generated by Bergenthum’s formulation tend to return process models with high fitness, uncontrolled precision and very unreadable by humans due to their complexity. That means, process models generated in this family of algorithms tend to represent well the observed



history, but they can represent many other things not observed in history.

Therefore, a sequence of research questions emerged during our research. The list of questions is:

- How to derive the whole Petri net from a single integer formulation? (Chapter 4)
- Is it possible to handle larger instances with an integer optimization approach in a time-efficient manner? (Chapter 5)
- How can we combine different algorithms into the ILP Miner approach? (Chapter 5)
- How can we trade-off the natural high replay-fitness yield by the ILP-based algorithms to obtain simpler and more precise process models? (Chapter 6)

## 1.2

### Structure of this work

This dissertation explores three major algorithmic ideas onto the standard ILP Miner algorithm. We present some theoretical background on the following two chapters, present each of the three ideas in their own experimental chapter, and discuss all results in a final chapter. Thus, we dedicate chapter 2 to delving into some common definitions and nomenclature, and some previous results in the literature that will be used later in this dissertation. Chapter 3 presents some theoretical discoveries and interpretations on the classic model which set the stage for developing the main algorithmic ideas in the following experimental chapters.

The subsequent chapters explore algorithmic advances with experimental setups. Chapter 4 presents the global formulation which allows getting a whole connected Petri net in a single run of the mathematical model. In this chapter, we test several versions of the formulation in toy problems that are intended to reproduce the conventional problems process discovery algorithms face on a small scale. This chapter serves as a proof-of-concept for further development in later chapters.

In Chapter 5 we present a way of decomposing the global integer model in a Branch-Cut-and-Price schema. The global compact model has to handle a large number of flow-inducing cuts and a large number of binary variables. Therefore, we propose a way of lazily introducing variables and flow inducing cuts to solve the optimization problem efficiently. We test this algorithm by mining instances present on the 2016 and 2017 Process discovery contest and

measuring the conformance of the models generating using measures of replay fitness and precision.

We observe in Chapter 5 that usually the models we obtain tend to have very high replay-fitness and very poor precision. This tendency means they represent the models can reproduce the observed event log, but also many other sequences not observed. Moreover, visually speaking, the models generated by the algorithm described in chapter 5 tend to be very hard for humans to understand, due to the complexity of connections.

Therefore, we present a way of navigating in this trade-off in Chapter 6. By giving up a small amount of replay fitness, we observed that it is possible to have a substantial gain in precision and simplicity (readability) in some of the instances. The downside is the large amount of extra computation required.

Finally, in Chapter 7, we discuss significant findings, limitations of the present work and future research. We believe that this approach is still vastly unexplored and could bring about powerful algorithms for the process discovery problem in the future.

## 2

## Basic concepts and definitions

### 2.1

#### Process mining

**Common notation** Let  $A$  be a set of elements, each element represents an activity or an event in a log. For instance, set  $A$  below contains activities  $a$ ,  $b$ ,  $c$  and  $d$ , i.e.  $A = \{a, b, c, d\}$ . A *sequence*, or a *trace* is a string of elements in  $A$  which describes the flow of a *case* throughout the business process. Each element in a trace is associated to the order of occurrence of each activity in the business process case. A sequence  $\sigma$ , as represented below, indicates that the activities  $a, b$  and  $d$  occurs in the following order  $d, a, b, a$  and  $d$ . Traces of a log are sequences of events, in this case  $\sigma = \langle d, a, b, a, d \rangle$ . Sequences can also be concatenated as in the following example.

$$\sigma_1 \cdot \sigma_2 = \langle d, a \rangle \cdot \langle b, a, d \rangle = \langle d, a, b, a, d \rangle$$

A set of sequences is a log:

$$X_1 = \{\langle a, b \rangle, \langle d, a, c \rangle, \langle b, b, c, a \rangle\}$$

The set of all possible sequences in the set  $A$  is represented by  $A^*$ .

**Bag-of-traces and prefix-closed Language:** A bag is different from a set since it attributes quantities for each element or sequence, according to whether it is a bag of elements or a bag of sequences. A bag  $B$  containing the sequences  $\langle a, b \rangle$  once,  $\langle d, a, c \rangle$  three times and  $\langle b, b, c, a \rangle$  four times is represented as follows:

$$B_1 = \{\langle a, b \rangle^1, \langle d, a, c \rangle^3, \langle b, b, c, a \rangle^4\}$$

A log of activities or an event log can be represented as a bag of sequences, each sequence being a trace and the quantity representing the number of times that trace occurs in the log. Given a set of sequences  $X$  its prefix-closure  $\overline{X}$  is the set of every sequence  $\sigma$  that belongs to  $X$  and all sequences  $\sigma_1$  that are a

prefix of  $\sigma$ , i.e.  $\sigma_1$  is such that  $\sigma_1 \cdot \sigma_2 = \sigma$ , we write:

$$\overline{X} = \{\sigma_1 \in A^* \mid \exists \sigma_2 \in A^* (\sigma_1 \cdot \sigma_2 \in X)\}$$

**Petri net:** We can define a Petri net as a bipartite directed graph. The two disjoint sets are  $T$ , the set of transitions, and  $P$ , the set of places. The set of arcs (or links)  $F$  is a subset of  $\{(T \times P) \cup (P \times T)\}$ . We denote this bipartite graph as  $G = (P, T, F)$ .

The simulation of a Petri net uses markings, i.e., tokens present at the places of the net. A marking  $m$  specifies the number of tokens at each place in  $P$ , and this configures a state of the Petri net. A Petri net at any given state can be entirely described by  $P, T, F$  and  $m$ ; a marked Petri net.

A transition  $t$  is enabled when all the places  $p$  that are predecessors of  $t$  in  $G$  have at least one token. As a result, the transition fires, and one token at each of  $t$ 's predecessors is consumed, generating an extra token at each of  $t$ 's successors.

**Firing sequence:** Given a marked Petri net  $P, T, F$  and  $m_0$ , a sequence  $\sigma$  over  $T$ , i.e.  $\sigma \in T^*$  is a firing sequence of this Petri net if there exists a sequence of markings  $m_1, m_2, \dots, m_{|\sigma|}$  such that for  $\sigma = \langle t_1, \dots, t_{|\sigma|} \rangle$ , transition  $t_j$  is enabled by marking  $m_{j-1}$  resulting in marking  $m_j$ , for  $j = 1, \dots, |\sigma|$ .

Observe that the set of transitions  $T$  of a Petri net corresponds to the set of activities/events  $A$  in the process discovery context. One firing sequence of a Petri net corresponds to a trace in an activity log. The set of all firing sequences of a Petri net, therefore, should correspond to the activity log of the business process it models. Metrics relating the original log and the one produced by the Petri net indicate how well it represents the business process.

## 2.2

### Linear Programming Formulation with Parikh vectors

The Process Discovery problem seeks a business model that corresponds to a meaningful representation of a process that generated a given log. We propose an integer linear programming (ILP) formulation that, given a log, outputs values to its variables that correspond to a Petri net. We search for Petri nets with a minimum number of links, connections between two activities, or a minimum number of places. We formalize the proposed ILP model for the Process Discovery problem.

**Input** An alphabet  $A$ , a set of sequences  $\mathcal{L}$  on elements of  $A$ .  $A$  contains two special symbols  $*$  and  $\#$  that correspond to the start and the end of all sequences in  $\mathcal{L}$ , respectively.  $*$  and  $\#$  only appear once in any sequence  $\sigma$  in  $\mathcal{L}$ .

**Output** A Petri net with a set of transitions  $A$  such that it produces approximately  $\mathcal{L}$ , minimizing some structural property of the Petri net (i.e., number of places, number of “links”).

In van der Werf (VanZelst2015), the author presents an integer linear programming (ILP) formulation that finds one place of a Petri net at each optimization step. This formulation captures candidate places of the Petri net as a basic solution of the optimization problem.

The method in van der Werf (VanDerWerf2009) then inserts new constraints that cut the solution space to push the solution of the integer program to a new optimal place. Iteratively, the algorithm finds places to construct the Petri net and terminates.

The ILP model in van de Werf et al. (VanDerWerf2009). It follows the use of Language-Based Theory of Regions in Bergethum et al. (Bergenthum2007) and in Lorenz and Juhás (Lorenz2007HowTS). While (Bergenthum2007) proposes algorithms that obtain several places through the resolution of an integer program and strives to eliminate redundant places, (VanDerWerf2009) makes use of objective functions and additional constraints that control this effect.

In a loose sense, the intuition behind these models is that adding a place, together with its connections, to a Petri net, reduces the set of sequences it generates. Therefore, previous ILP based approaches solve several integer programs in order to obtain places that assemble the desired Petri net. We proceed in this section presenting the formulation for a single place, as in (VanDerWerf2009), (Bergenthum2007).

## 2.3

### Base formulation for a single place

The way to represent a region is through determining the activities with arcs entering and leaving the region. Since a region cannot block any sequence of the language defined by the log, each prefix of each sequence is used to determine the base constraints of the model. Let, for  $q \in A$ ,  $y_q$  be binary variables that indicate there is an arc leaving the region to activity  $q$  and let  $x_q$  indicates that the region has an arc leaving to activity  $q$ . The sequence

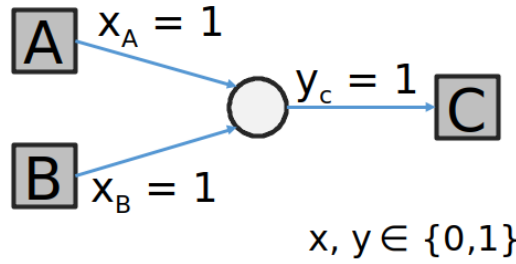


Figure 2.1: An example of the solution. We represent with the variables  $x_i$  and  $y_j$  the presence or absence of an arc incoming or going to a transition respectively

$\langle a, b, c \rangle$  in the activity log implies constraints:

$$\begin{array}{rcl|l}
 x_* - y_a & \geq & 0 & \langle a \rangle \\
 x_* - y_a + x_a - y_b & \geq & 0 & \langle a, b \rangle \\
 x_* - y_a + x_a - y_b + x_b - y_c & \geq & 0 & \langle a, b, c \rangle
 \end{array}$$

Suppose that, for each trace, for each prefix of this trace  $s$ , the sum of arcs that goes out of the prefix to a Petri net place must be at least the number of arcs that come from a Petri net place to  $s$ . The main constraint set of ILP formulations generating one place at a time is the one above. We build our complete Petri net formulation on this constraint set. The rest of this work will refer to this formulation as the base formulation, classic formulation or Bergenthum's formulation.

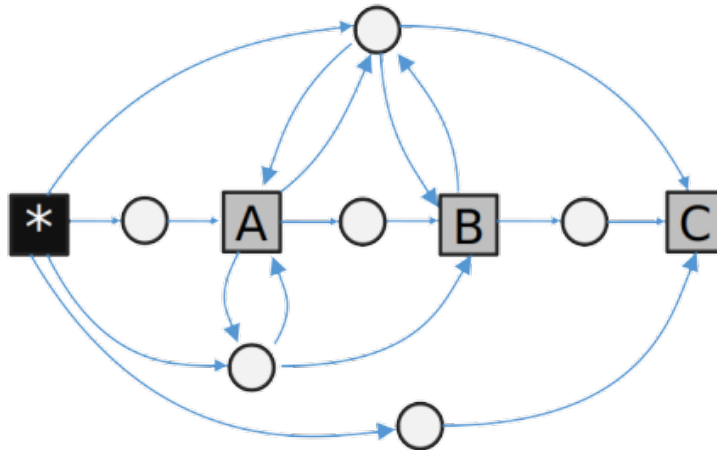


Figure 2.2: Viable solutions of the classic formulation using a single sequence language  $\langle A, B, C \rangle$ . Observe that there are many places that satisfy the constraints from the model. Also observe that imposing a minimal number of arcs yield the most simple place candidates.

Formally, we can define that an observed event log  $X$  is enlarged to a prefix-closed language  $\overline{X}$ . We can represent this operation with a line above

the variable that represents the set.

$$\overline{X} = \{\sigma_1 \in A^* \mid \exists \sigma_2 \in A^* (\sigma_1 \cdot \sigma_2 \in X)\}$$

In words, the prefix closure operation over an event log  $X$  creates a set in which every element  $\sigma_1$  is a sequence made from symbols of the alphabet  $A$  that when concatenated with another sequence  $\sigma_2$  made from symbols of the alphabet  $A$  generates a sequence already in the event log  $X$ .

Each of these prefixes  $\sigma_1$  present in  $\overline{X}$  will originate an inequation over binary variables  $x_i$  and  $y_j$  that represent incoming and outgoing arcs of a solution place.

$$x_*^k - y_q^k + \sum_{j \in \sigma_1} (x_j^k - y_j^k) \geq 0 \quad \forall \sigma_1, q \mid \exists \sigma \in S \text{ and } \sigma_2, \sigma = *. \sigma_1. q. \sigma_2, \quad \forall k \quad (2-1)$$

Notice that this formulation obtains a single feasible place. Usually, algorithms tend to enumerate feasible solutions to this model. Later, algorithm designers need to implement a strategy to filter and to guarantee cohesion between this list of places.

Our intention of proposing this new formulation is to take advantage of the power and flexibility of modeling problems in terms of mathematical programming and obtain the whole Petri net using a formulation.

We believe that a formulation that addresses the whole process may be interesting to bring options for a process modeler to control global properties of a Petri net such as token throughput and cohesion between places.

## 2.4

### Challenges of measuring conformance

Measuring objectively how good is a model remains an open question in the field. It is possible to obtain process models that have artificially good characteristics, but in practice do not encode process characteristics such as flow diversity, loops and kinds of decisions.

Common metrics used in the field are precision and replay fitness. Replay fitness is closely related to the concept of recall in other fields. A model is fit whenever it can reproduce a large amount of the behavior observed in history. Precision, as a complementary metric, should express how economical our model is. Of all possible behavior produced by the model, how much it is in the observed history. As opposed to fitness, a model has low precision if it expresses many other behaviors not expressed in history. A model is precise if it encodes only behavior observed in history.

Even though these concepts seem enough to derive sufficient metrics to determine whether models are good, they are not. To illustrate this, we will present two examples of ways of obtaining models that in a way cheat objective metrics.

The first one is the star model. This model obtains high fitness because it can express any sequence of the activities in its substructure in any quantity and any order. Unless the log has all possible permutations of a given sequence, this model should have very low precision, because it can reproduce all possible sequences using the alphabet.

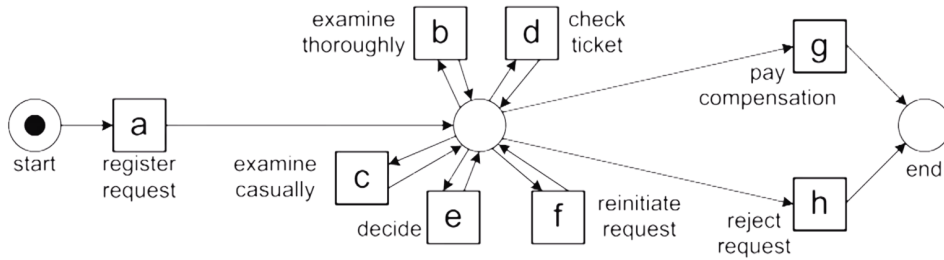


Figure 2.3: An example of the star model from (Rozinat2007). Every sequence of every length is possible using the transitions that have incoming and ongoing arcs to the same place.

Another way of exploiting these metrics unfairly is to build linear Petri nets according to the given sequences. Then we put at the start and finish of the model an place to produce an effect of exclusive choice. Notice that this model will represent correctly the model and only the model. Therefore, it does not matter which metric we use to measure its fitness and precision; it should score high.

According to Aalst in (VanDerAalst2016), an acceptable process model should then balance four aspects of the process discovery task: fitness, precision, simplicity, and generalization. Simplicity is defined as how complicated is a model. If the model has a graphical representation, a simple model would use a few connections and would have a streamlined morphology instead of entangled connections.

In this work we will use in Chapter 4 a simple binary go/no-go indicator for toy problems to which we have the expected Petri nets. However, we deal with complex logs in Chapters 5 and 6, and the binary indicator is no longer meaningful. Therefore, we present in Chapter 5 ways of measuring fitness and precision detailed in (VanDerAalst2016), (Adriansyah2015), (Munoz2010), (Carmona2018).



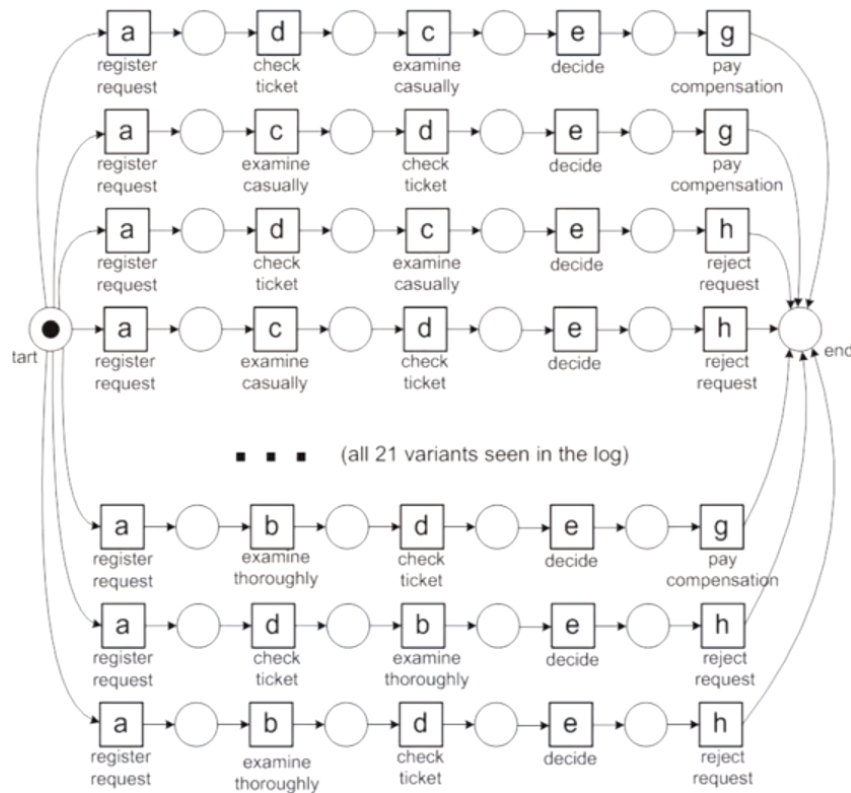


Figure 2.4: An example of a linear enumeration model from (Rozinat2007) . We can simply build linear models using the language and an exclusive choice at the beginning to achieve maximum fitness and maximum precision. We have also to take into consideration the simplicity of the model as a function of the number of arcs and transitions of the same symbol used.

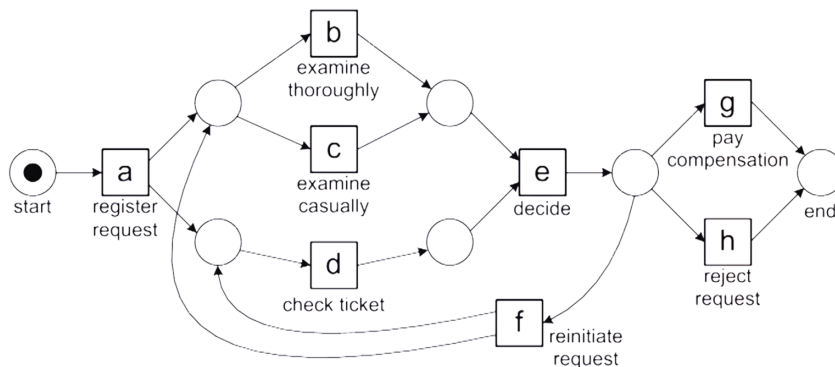


Figure 2.5: An example of a balanced petri net from (Rozinat2007). It does not achieve maximum fitness or precision, but it is simple. The consequence of balancing is a model in which the connections denote parallelism, decisions and other behaviors useful for a business process analyst clearly.

### 3

## Analysis and interpretation of the classic formulation

This section is about the enabling theoretical aspects of our approach. The classic formulation has a set of inequalities which describe place candidates as the basic solutions. We first show that the left-hand side coefficient matrix is Totally Unimodular. Consequently, any basic solution to this set of inequalities can be solved by any linear solvers with a guarantee of finding an integer solution given an integer right-hand side and integer objective.

Then the problem of finding one place candidate falls into a particular category in which can be solved efficiently. Usually, problems with the unimodularity property have deep connections to other combinatorial optimization problems. Hence this may be a new path to building reliable heuristics to the process discovery problem.

Additionally, recent advances in other algorithms may have implications for the ILP's classic formulation. For instance, in multiple string comparison and in outside-in process discovery algorithms, it is usual to use an auxiliary graph, called directly-follows. Algorithm designers use this graph to prove that there are tightly connected subsets of the alphabet which could be solved separately.

In the end, we present some intuition on how ILP is susceptible to outlier logs. Also, we conjecture a search for places which do not attend

### 3.1

#### Unimodularity

First, we prove that the left-hand side matrix of the classic formulation using the prefix closed language Parikh vectors is totally unimodular. Total unimodularity is an important property for Integer Programming, because it is possible to use linear solvers to obtain integer solutions, given that coefficients in the right-hand side are integers. Therefore, the problem of finding one place candidate reduces to  $P$ .

As an overview, in the book (Conforti2014), Conforti, Cornuéjols and Zambelli prove and enlist a set of matrix operations which preserve total unimodularity. Therefore, if a classic total unimodular matrix, such as the node-

arc matrix description of a directed graph, could be obtained by unimodularity preserving operations, this matrix would also be totally unimodular.

Then, the formulation is totally unimodular because the it is defined from a prefix-closed set. Using the immediate prefix, we always can find a way of obtaining through unimodularity preserving operations a typical bipartite flow matrix which is totally unimodular.

**Lemma 3.1 (Unimodularity preserving matrix operations)** *The following matrix operations, called unimodular operations, preserve the unimodularity property from an already unimodular matrix:*

1. *Interchange two columns*
2. *Add an integer multiple of a column to another column*
3. *Multiply a column by  $-1$*
4. *Transpose*

*Proof.* For items, 1 through 3 please see (Conforti2014). For item 4, by the definition of totally unimodular matrices, all determinants of non-singular square submatrices must be  $\in \{-1, 0, 1\}$ . The transposition of a matrix also transposes all submatrices. Additionally, transposition does not affect the determinant of a matrix. Therefore, if all determinants of submatrices were already  $\in \{-1, 0, 1\}$ , they will not change by the transposition, and the matrix will remain Totally Unimodular. ■

**Corollary 3.2** *Since transposing preserves unimodularity, we can also perform the same operations described for columns and apply them to rows preserving the matrix unimodularity, by concatenating transposition operations before and after column operations.*

**Lemma 3.3 (From Identity to Unimodularity)** *Matrices obtained from the identity through a series of unimodular operations are also unimodular.*

*Proof.* Once more we refer to the first chapter of Conforti and Conrnuéjol's book (Conforti2014). ■

For example, the encoding of a directed bipartite graph in a node-arc fashion is always totally unimodular, because one can always find a way to exploit the particular structure of a matrix to arrive at an Identity concatenated with a matrix of zeros.

**Theorem 3.4 (ILP Formulation's Total Unimodularity)** *Any matrix composed of the application of a Parikh vector function over a prefix-closed set of sequences (or Language) set is totally Unimodular.*

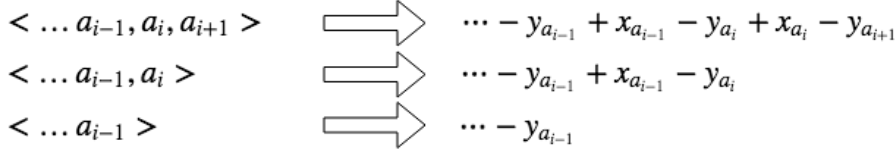


Figure 3.1: Prefixes transform into the left-hand side of the constraint inequalities. If the set is a closure of prefixes, we can guarantee that for each sequence with length above one symbol will have its immediate prefix within the closure. By immediate prefix we mean the sequence with just the last symbol missing of a giving sequence. Thus, if the last symbol of the sequence  $\sigma_1$  is  $a_{i+1}$  and the last symbol of its immediate prefix sequence  $\sigma_2$  is  $a_i$ , we expect the left-hand side of the inequality associated to  $\sigma_1$  be the same as the inequality associated to  $\sigma_2$  added  $+x_{a_i} - y_{a_{i+1}}$ .

*Proof.* Each observed variant (VanDerAalst2013) in history generates a set of prefixes, and each prefix of this set originates an inequality. The union of all set of prefixes generates a prefix-closed language  $\bar{L}$ . With this property, for all non-single sequences in the Language  $\bar{L}$  there is an immediate prefix present on the set.

The sequence  $\sigma_{ip}$  is the immediate prefix of a given original sequence  $\sigma_o$  if it just needs a concatenation with just one element  $a$  to equal  $\sigma_o$ .

$$\text{immediate\_prefix}(\sigma_o) = \sigma_{ip} \iff \sigma_{ip} \cdot \langle a \rangle = \sigma_o$$

Therefore, aside from sequences composed of the artificially inserted beginning element and one valid element, the immediate prefix of every sequence of the language is also an element in the set.

Then, if we subtract the rows of coefficients of between any sequence and their immediate prefix, we get a row composed by all zeros and one entries 1 and other -1. The transpose of this resulting matrix always has the particular form of a node-arc incidence of a bipartite graph which is totally unimodular.

■

Furthermore, after the procedure, columns related to the  $y$  variables have only 0 or -1 entries, whereas columns related to the  $x$  variables have only 0 or 1. This fact suggests that the transpose is not only a node-arc incidence matrix of a regular graph, it is of a bipartite graph.

To conclude with an example, consider the log  $L = \langle A, B, C \rangle, \langle A, D, C \rangle$ . First, we add the artificial events of start ' $*$ ' and finish ' $\#$ ' and build the prefix closed language set of sequences  $\bar{L}$ .

We encode  $\bar{L}$  as a matrix by computing in each row the parikh vector for each sequence. For instance, the sequence/prefix  $\langle *, A, B \rangle$  is associated with the inequality  $x_* - y_A + x_A - y_B \geq 0$ .

With the alphabet (set of all possible activities)  $\mathcal{A} = \{A, B, C, D\}$  the variables created are  $x_* - y_A + x_A - y_B + x_B - y_C + x_C - y_D + x_D - y_\#$ . If we establish this as an ordering of variables, we can use it to define a Parikh function  $\vec{p}(\sigma)$  to encode the prefixes as a vector of coefficients (Parikh vector).

Finally, the sequence  $\langle *, A, B \rangle$  produces the row  $[1, -1, 1, -1, 0, 0, 0, 0, 0]$ . Alternatively, formally:

$$\vec{p}(\langle *, A, B \rangle) = [1, -1, 1, -1, 0, 0, 0, 0, 0]$$

The prefix-closed language is guaranteed to have the immediate prefix of every non-trivial sequence always. Therefore, if we observe  $\langle *, A, B \rangle$ , there must be  $\langle *, A \rangle$  in  $\bar{L}$  as well. Additionally:

$$\vec{p}(\langle *, A \rangle) = [1, -1, 0, 0, 0, 0, 0, 0, 0]$$

and finally:

$$\vec{p}(\langle *, A, B \rangle) - \vec{p}(\langle *, A \rangle) = [0, 0, 1, -1, 0, 0, 0, 0, 0]$$

Thus, we always can operate row subtractions and additions to arrive at the transpose of a typical directed node-arc incidence matrix. Then, this formulation is Totally Unimodular.

Below, we present a full example of achieving the transpose of a directed node-arc incidence matrix.

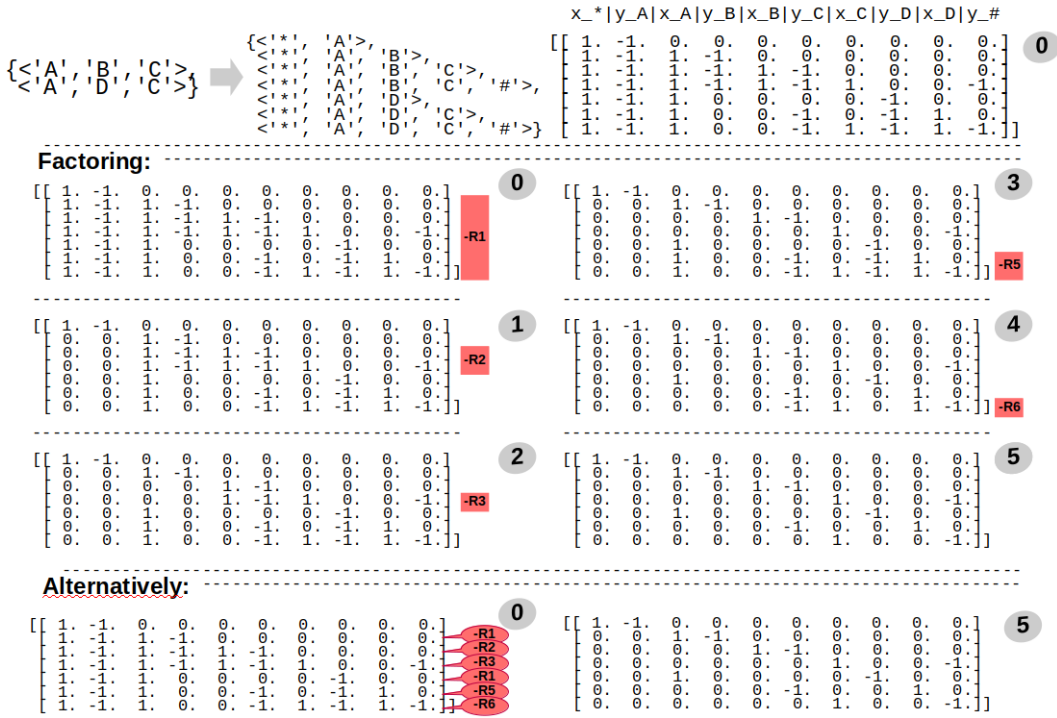


Figure 3.2: Many heuristics could be used to factor the left-hand side matrix of the formulation. The proof we have given based on immediate prefixes is the alternative example at the bottom of this picture.

In the example we suggest two paths to go from matrix 0 to the matrix 5. The matrix 0 is the coefficient matrix that we want to prove is Totally Unimodular. Matrix 5 is the transpose of a directed node-arc incidence matrix, which is known to be Totally Unimodular.

### 3.2

#### Alphabet filtering

All constraints of the classic formulation enforce that the Parikh function over  $X$  and  $Y$  variables must be at least zero. Suppose a situation where the process analyst has reason to believe that places are using only a sub-alphabet  $\mathcal{A}'$ , a subset of the alphabet  $\mathcal{A}$ .

He or she could fix all variables related to events not in  $\mathcal{A}'$  to zero and re-execute the solver to generate a place. Once the solver generates a new place, it will attribute to variables related to the events in  $\mathcal{A}'$  values 0 or 1. Since the constraints over these variables enforce them to be greater than zero, if the solver finds an optimal solution to this new problem with fixed variables, this solution would also be feasible in the original formulation.

Alternatively, it is possible to build a new optimization model only filtering the event log, creating what we will call sublog. For instance, imagine the event log:  $\{ \langle A, B, C, D \rangle, \langle A, C, B, D \rangle \}$ , imagine that we want to

focus on just the events  $C$  and  $D$ . Thus we filter the event log into the sublog:  $\{< B, C >, < C, B >\}$ . Now please observe what happens if we construct the formulation for both the given log and the resulting filtered sublog:

For the event log  $\{< A, B, C, D >, < A, C, B, D >\}$ :

$$\begin{array}{rcll}
 x_* - y_A & \geq 0 & | & < A > \\
 x_* - y_A + x_A - y_B & \geq 0 & | & < A, B > \\
 x_* - y_A + x_A - y_B + x_B - y_C & \geq 0 & | & < A, B, C > \\
 x_* - y_A + x_A - y_B + x_B - y_C + x_C - y_D & \geq 0 & | & < A, B, C, D > \\
 x_* - y_A + x_A - y_C & \geq 0 & | & < A, C > \\
 x_* - y_A + x_A - y_B - y_C + x_C & \geq 0 & | & < A, C, B > \\
 x_* - y_A + x_A - y_B + x_B - y_C + x_C - y_D & \geq 0 & | & < A, C, B, D >
 \end{array}$$

For the filtered sublog:  $\{< B, C >, < C, B >\}$

$$\begin{array}{rcll}
 x_* & \geq 0 & | & < > \\
 x_* - y_B & \geq 0 & | & < B > \\
 x_* - y_B + x_B - y_C & \geq 0 & | & < B, C > \\
 x_* - y_C & \geq 0 & | & < C > \\
 x_* - y_B - y_C + x_C & \geq 0 & | & < C, B >
 \end{array}$$

Both set of inequalities are equivalent if we fix  $y_A = x_A = y_D = 0$ , which are the variables related to the events  $A$  and  $D$  outside the subalphabet  $\mathcal{A}'$ .

**Theorem 3.5** *The basic solutions of the classic formulation applied to unfiltered log contains the set of all basic solutions of the classic formulation applied to the filtered sublogs.*

*Proof.* By construction, the resulting set of inequalities from the sublog have the same coefficients for the variables associated with events in the sub-alphabet.

As argued before, fixating the variables related to events which are not in the subalphabet  $\mathcal{A}'$  to zero, will have a similar effect of not optimizing them at all. The constraints enforce the sum to be more than zero. If they are more than zero using fewer variables, they will also be feasible if we fixate the variables not considered to zero.

Therefore, if we remove the variables, not in the subalphabet and call the solver for the remaining variables associated with the sub-alphabet, we can reinsert this solution into the original problem and fill the remaining variables with the value zero and get a feasible basis in the original subproblem.

Therefore, if all basis in this reduced problem with a subalphabet can be inserted into the main program to obtain a feasible basis, then all basis in the

filtered subproblem are translatable to a feasible basis in the original problem.

■

Then it is possible as a heuristic to create alternative place generators using the classic formulation over a filtered set of prefix-closure. We can benefit from the reduced amount of integer variables needed.

### 3.3

#### Susceptibility to outlier logs

The original set of inequalities express the constraints that a place must comply in order to respect the observed history. We transform the set of inequalities into a linear program forcing the solver to return a place without any nonessential arcs.

In a real setting, capturing the data or the process itself is subject to abnormalities. The observations may be affected from time to time. For instance, an unobserved sequence in the past may occur, or a sequence observed may not occur again.

In an algebraic sense, one translate each observed sequence of activities to an inequality which describes a semi-space. Considering all generated inequalities, we have the intersection of all semi-spaces which is a polyhedron. In each vertex of these polyhedra lie the basic solutions for the set of inequalities, representing valid place candidates. Since the problem is unimodular, its relaxation always has integer basis for a vector of zeros in the right-hand side.

In the formulation, such unobserved event sequence would generate a new set of inequalities, consequently could "cut" out of the space of feasible places, a lot of good candidates, whereas the absence of a sequence, could mean the relaxation of the related inequalities and augmenting the space of feasible places.

In other words, this polyhedron could be enlarged or shortened, and therefore the catalog of valid places will vary accordingly. Therefore the set of feasible places available to form the Petri net would change according to observations in each point in time.

One common way to mitigate these effects is to filter out infrequent behavior. This approach tries to reach one unique polyhedron which would be static throughout time. It supposes that the sequences observed at least a certain number of times would always be observed again whereas the infrequent logs would deviate freely without entering the analysis. The problem with this approach is that it does not take into consideration similar behavior among infrequent logs. The case in which all traces occurred just once is an example in which this strategy would not succeed.



An alternative would be to enlarge the set of places taken into consideration. We could build multiple polyhedra, considering different sets of sequences. In other words, from the original polyhedron, we could create larger polyhedrons by selectively turning on and off constraints related to infrequent logs and consider all the places relative to all vertices in these resulting polyhedra.

For instance, we could build all sets of inequalities in which describe for than  $X\%$  of the history. Therefore, each of these polyhedra formed by different combinations of the original inequalities would form a different scenario representing a different manifestation of the sequences of events. This concept is at the heart of the risk-prone model that we show in section 6 of this dissertation.

## 4

### Global Integer Programming - the compact model

The goal of this section is to propose an integer linear program that returns a full Petri net representation of the business process as output. The initial idea is to build a direct network from event to event while trying to cover this network using places generated by the classic formulation presented before. To control this covering, we must translate the places generated into bicliques of symbols.

#### 4.1

##### The global model

We devise the proposed model applying the following three building blocks: (1) replication of the basic program for an arbitrary number of candidate places, (2) creation of integer variables to allow or prohibit flow between transitions of the Petri net, (3) force flow through the Petri net, from the dummy start transition  $*$  to the dummy ending transition  $\#$ .

The model we devise finds simultaneously several places while imposing constraints on the resulting Petri net, i.e., the one induced by the set of places obtained. Mainly, these constraints say that the Petri net must correspond to a connected graph and that the Petri net should balance the generation and the consumption of tokens.

The way the model address the problem of drawing the full Petri net, finding all the places, is considering up to  $K$  candidate places and  $K$  input and  $K$  output variables for each activity. In a log with activities  $a, b, c$ , these variables are  $x_a^k, x_b^k, x_c^k, y_a^k, y_b^k$  and  $y_c^k$ , being  $K$  a hyper-parameter. The upper bound on the number  $K$  of candidate places is arbitrarily defined. We then replicate the base formulation for these  $K$  places. Additional variables and constraints are then introduced to force cohesion among the  $K$  places.

The model has four variables sets. Variables  $x_j^k$  and  $y_j^k$  for  $j \in A$  and  $k = 1, \dots, K$ , as described above, and variables  $z_{i,j}^k$ , with  $k = 1, \dots, K$ , and  $w_{i,j}$ , for all pairs  $(i, j)$  of activities. While variables  $a_{i,j}^k$  represent a directed connection between two activities  $i, j$  by the place  $k$ , variables  $w_{i,j}$  are set to one whenever a place  $k$  connects the pair of activities  $(i, j)$ .

**Trace constraints:** Correspond to the constraints for obtaining a place replicated for  $K$  candidate places. It writes:

$$x_*^k - y_q^k + \sum_{j \in \sigma_1} (x_j^k - y_j^k) \geq 0 \quad \forall \sigma_1, q \mid \exists \sigma \in S \text{ and } \sigma_2, \sigma = *. \sigma_1. q. . \sigma_2, \quad \forall k \quad (4-1)$$

where  $q$  is the last activity in the prefix considered of the sequence in the log.

**Transition connectivity:** A connection linking transition  $i$  to transition  $j$  is induced by place  $k$  when both  $x_i^k$  and  $y_j^k$  are set to one. In other words, Activity  $i$  has an arc leaving to place  $k$ , which has an arc leaving to activity  $j$ . The resulting constraints can be written:

$$a_{ij}^k \leq x_i^k \quad \forall (i, j) \in A \times A, k = 1, \dots, K \quad (4-2)$$

$$a_{ij}^k \leq y_j^k \quad \forall (i, j) \in A \times A, k = 1, \dots, K \quad (4-3)$$

$$a_{ij}^k \geq x_i^k + y_j^k - 1 \quad \forall (i, j) \in A \times A, k = 1, \dots, K \quad (4-4)$$

Variables  $w_{ij}$  indicate the Petri net contains a link from activity  $i$  to  $j$ , this is true if at least one place induces this link. The constraints below link variables  $a$  with variables  $w$ .

- If the Petri net links  $i$  to  $j$ , then at least one place must induce this connection:

$$\sum_{k=1}^K a_{ij}^k \geq w_{ij} \quad \forall (i, j) \in A \times A \quad (4-5)$$

- If at least one place induces the link from  $i$  to  $j$ , then the Petri net has this link:

$$w_{ij} \geq a_{ij}^k \quad \forall (i, j) \in A \times A \wedge \forall k \in K \quad (4-6)$$

There is a more general interpretation of this formulation. The blocks of  $X$  and  $Y$  variables describe respectively incoming arcs to places  $k$  and outgoing arcs from place  $k$ . The original arcs either connect events to candidate place  $k$  or the other way.

The global program creates a family of arcs that link events directly to other events. This new network has a different nature from the original arcs. Arcs of the Petri net can either connect a transition to a place or a place to a transition. However, at this point in the execution of the algorithm, we can only enumerate transitions linked to the types of events in the alphabet.

To connect the two worlds of arcs  $W$  and the original  $X$  and  $Y$ , we propose a translation variable  $A$ . The idea behind  $A$  is to transform the feasible candidate place described in  $X$  and  $Y$  to direct connections. For instance if place  $k = 3$  has  $x_A^3 = x_B^3 = y_C^3 = y_D^3 = 1$  and the remaining  $X^3$  and  $Y^3$  equal

to zero, then place 3 has incoming arcs from events A and B and outgoing arcs to event C and D.

In this example, we translate the  $X$  and  $Y$  variables to  $A$  in the following manner:  $A_{AC}^3 = A_{AD}^3 = A_{BC}^3 = A_{BD}^3 = 1$ . The  $A_{ij}^k$  variables represent the multiplication of  $X_i^k$  and  $Y_j^k$  or if place  $k$  connects events  $i$  and  $j$ . Therefore we can use the presented Fortet linearized multiplication (Fortet1960).

We translate the place to a biclique. In the former example, the place 3 is a biclique from the bipartite  $(A, B) \rightarrow (C, D)$  and the variables  $A$  denote the arcs from event to event present in this biclique.

Finally, using all the  $K$  candidate places generated this way, we want to cover the graph of  $W$  variables using the bicliques described by the  $A$  variables.

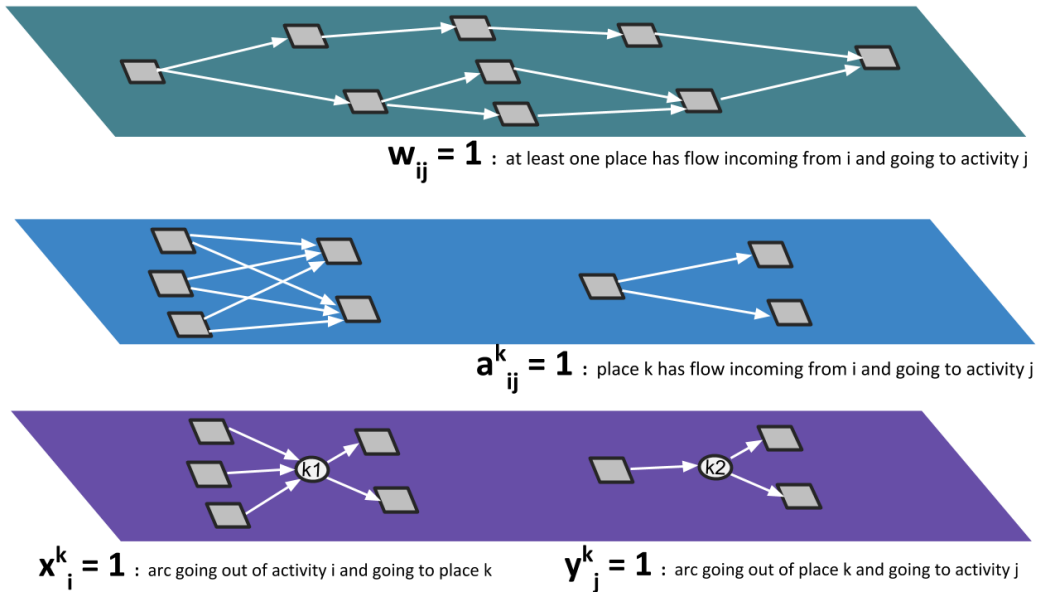


Figure 4.1: Graphical representation of the layers imagined for the global formulation. Here the original  $x_i^k$  and  $y_j^k$  variables are depicted as a lower level in which we search for a place  $k$ . The upper layer is a direct network between events, connecting directly an event  $i$  to another  $j$  represented by  $w_{ij}$  variables. Then there is the middle layer in which translates  $x_i^k$  and  $y_j^k$  into a direct connection  $a_{ij}^k$  between transition  $i$  that goes to place  $k$  and transitions  $j$  that arrive from place  $k$ . The problem becomes to cover all  $w_{ij}$  using available  $a_{ij}^k$ .

**Petri net connectivity:** Let  $G = (A, W)$  be the support graph of a Petri net where the set of activities  $A$  is the vertex set and  $W = \{(i, j) | w_{ij} = 1\}$  is the arc set. Since all traces in the log start with activity  $*$  and ends with activity  $\#$ , and all activities in set  $A$  appear at least in one trace of the log, the supporting graph of the Petri net  $G = (A, W)$  must be connected. In particular,  $G$  must have at least one path from  $*$  to  $\#$  passing through every other activity in

A. This graph connectivity addresses the Petri net soundness as defined by (VanDerAalst2016).

We formulate these directed connectivity requirements of the Petri net supporting graph by imposing the any cut defined by  $D \subset A$  of the graph containing activity  $*$  and not containing  $\#$  must have at least one arc leaving. Alternatively, cuts for  $D$  containing activity  $\#$  and not containing  $*$  must have at least one entering arc. These cut constraints can be written:

$$\sum_{(i \in D)} \sum_{(j \in A-D)} w_{ij} \geq 1 \quad \forall D \subset A, * \in D \quad (4-7)$$

It consists of an objective function subject to constraints (4-1) - (4-7). Two straightforward objective functions lead to minimal Petri nets: minimize the number of links and minimize the number of places or arcs used by the places. Our tests balance those elements assigning weights to the use of the variables. Therefore:

$$GILP : \left\{ \begin{array}{l} \min W_1 \sum_{(i,j) \in A \times A} w_{ij} + W_2 \sum_{k=1}^K \sum_{i \in A} (x_i^k + y_i^k) \\ s.t. \\ (4-1) - (4-7) \\ x, y, z, w \text{ binary} \end{array} \right.$$

## 4.2

### Global Formulation's auxiliary ideas

In the previous sections the core ideas and the basic GILP formulation were presented, in this section auxiliary ideas are presented, some of them correspond to heuristic elements to improve the solution of the model. The auxiliary ideas are:

- Use trace suffixes in addition to trace prefixes
- Parallelism prohibition
- Fixing of starting and finish activities through  $w_{ij}$
- Imposing token handling symmetry

**Trace suffixes:** As supplementary constraint, we may consider the suffix instead of the prefix. Its validity is based on the same arguments for the prefix constraints from the language based theory of regions. The trace  $\langle a, b, c \rangle$  induces the following set of suffix constraints:

$$\begin{array}{rcl}
x_c^k - y_{\#}^k & \geq & 0 \\
x_b^k - y_c^k + x_c^k - y_{\#}^k & \geq & 0 \\
x_a^k - y_b^k + x_b^k - y_c^k + x_c^k - y_{\#}^k & \geq & 0
\end{array} \quad \left| \quad \begin{array}{l} < c > \\ < b, c > \\ < a, b, c > \end{array} \right. \quad (4-8)$$

While prefix constraints enforce places that connect transitions that are enforced by the beginning of the traces, suffix constraints enforce places that are induced by the end of the traces. Intuitively, assuming the all traces describe expected behaviors along the business process they should be redundant. On the other hand, when there are traces with anomalies in the log the use of both of these constraint types are expected to allow the methodology to find Petri nets.

**Non-related constraints:** Consider a pair of activities appearing in sequence in a trace of the activity log. We count the number of appearances of each possible ordered pair of activities. Let  $f$  denote this counting function, i.e.:

$$f : (A \times A) \rightarrow \mathbb{N} \quad (4-9)$$

Three types of possible activity relation can be extracted from  $f$ :

- Parallel Activities - Says that if a pair of activities  $i, j$  appear consecutively in a trace as  $ij$  and as  $ji$ , then there should not be a connection from  $i$  to  $j$  and neither from  $j$  to  $i$ .

$$\mathcal{P} = \{(i, j) | f(i, j) > 0 \wedge f(j, i) > 0\}$$

- Non related Activities - Says that if a pair of activities (transitions)  $i, j$  never appear consecutively in the whole log, there should not be a connection from  $i$  to  $j$ .

$$\neg\mathcal{R} = \{(i, j) | f(i, j) = 0 \wedge f(j, i) = 0\}$$

These sets are discussed thoroughly in van der Aalst(2016) (VanDerAalst2016). They are the basic elements for the alpha miner algorithm in the process discovery theory. In this work, the non-relating and the parallel set of activities are used to fix the initial status of the integer program variables. We configure them as hyperparameters to better understand the impact in the ILP resolution time and the Petri net output of the proposed integer programming model. In other words, we set:

$$w_{ij} = 0 \quad \forall (i, j) \in (\mathcal{P} \cup \neg\mathcal{R})$$

**Force start and end:** The proposed ILP model also allows fixing the initial and the final activities appearing in the traces of the activity log. We do so by setting variable  $w_{*i}$  to one if  $i$  is the first activity of some trace in the event log. Also,  $w_{i\#}$  is set to one if  $i$  is the last activity of some trace in the event log. We define set  $\mathcal{S}$  as the set containing all activities that start a trace and the analogous set  $\mathcal{E}$  containing all activities ending traces. Conversely, we set to zero the  $w$  variables associated to activities that in no trace appears as first activity or last activity. Therefore:

$$\begin{array}{llll} w_{*i} = 1 & \forall i \in \mathcal{S}; & w_{*i} = 0 & \forall i \notin \mathcal{S} \\ w_{i\#} = 1 & \forall i \in \mathcal{E}; & w_{i\#} = 0 & \forall i \notin \mathcal{E} \end{array}$$

**Token Symmetry:** Another constraint we use to increase the chance of producing sound Petri nets. We refer to the symmetry constraint that forces that the numbers of arcs going out of the overall set of places to be equal to the number of arcs arriving in the same place. We may write:

$$\sum_{k=1}^K \sum_{i \in A} x_i^k = \sum_{k=1}^K \sum_{i \in A} y_i^k \quad (4-10)$$

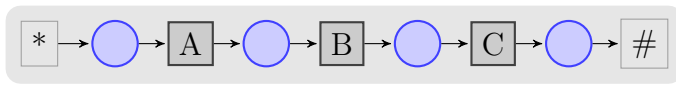
As mentioned above, the constraints described in this section may or may not be part of the formulation. Their use is to provide elements to be activated when searching for a sharper and more efficient formulation.

### 4.3 Global ILP Experiments

We demonstrate the behavior of the proposed ILP formulation as a discovering processes tool under several conditions. Our experiment uses 14 artificial and small logs that reproduce some common issues for process discovery algorithms, in particular for those which rely on finding places of the resulting Petri net. One test log is a simple sequence of activities; other has a decision; another contains a cycle. Also, we test the GILP model using nine experiment setups, defined by different uses of the constraints from the auxiliary ideas.

We devised 14 small logs to reproduce common situations faced by process discovery algorithms. Their heterogeneity allowed us to observe the strengths and weaknesses of this new method.

**Log 1 - Linear:** Simple linear process. First, we provide a simple baseline to observe the algorithm's effectiveness.

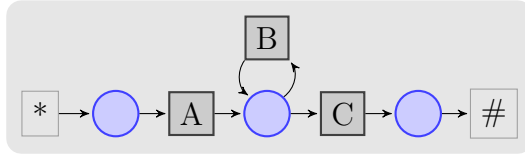


Log 1:

–  $\langle A, B, C \rangle$ 

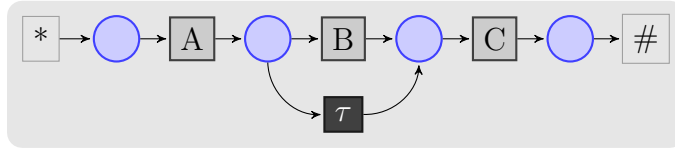
**Log 2 - Skip/loop:** Process with a skip or optional simple loop. This example introduces one of the first difficulties in modeling. Once we work with a few different traces, we may model this process as a single loop or as a skip.

Our formulation does not handle  $\tau$  transitions yet. But it is still interesting to observe the answers that each tested variant can return. We present different forms of modeling the log. Logs with skips and cycles have a variety of ways of representing, depending on the level of fidelity and readability required.



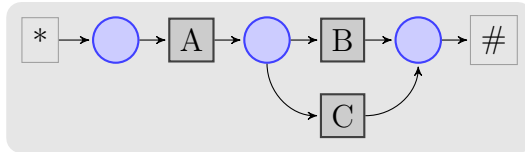
Log 2:

–  $\langle A, B, C \rangle$ ,  
–  $\langle A, C \rangle$



**Log 3 - OR-open:** Process with decision without closing. This log tests the ability to capture OR-gates without closing the flow.

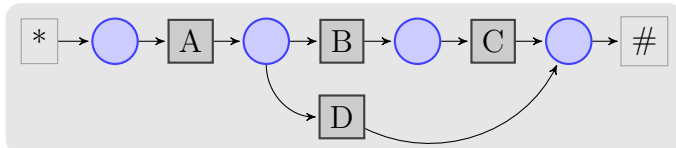
Log 3:



–  $\langle A, B \rangle$ ,  
–  $\langle A, C \rangle$

**Log 4 - asym. OR-open** Process with decision without closing 2. This log tests the capability of capturing OR-gates without closing the flow with a slight asymmetry.

Log 4:

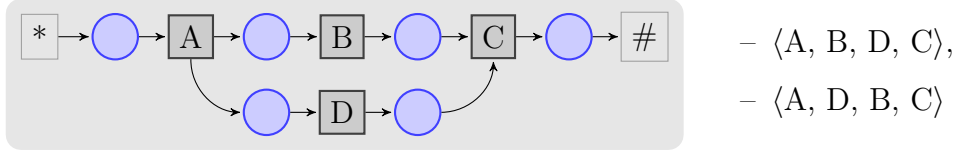


–  $\langle A, B, C \rangle$ ,  
–  $\langle A, D \rangle$



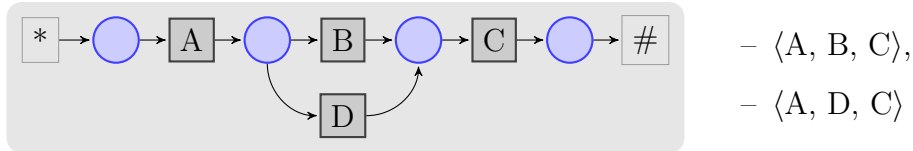
**Log 5 - AND-open/close:** Process with parallelism. In this log, we test the algorithm's capability of identifying all the multiple places that represent parallel flows. Traces in the log vary between themselves by changing the order of activities. Therefore, the algorithm must be capable of identifying parallel and consecutive activities to form different streams.

Log 5:



**Log 6 - OR-open/close:** Process with decision with closing. Here, we test whether the model can capture a pair of OR-gates used for opening and closing different streams.

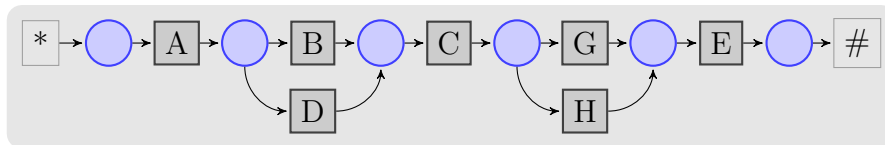
Log 6:



**Log 7 - 2OR-open/close:** Process with two decisions with closing. This log scales up the previous log testing decisions to two consecutive decisions. This model contains four places which have more than one entering arc or more than one leaving arc.

Log 7:

- $\langle A, B, C, G, E \rangle$ ,
- $\langle A, B, C, H, E \rangle$ ,
- $\langle A, D, C, G, E \rangle$ ,
- $\langle A, D, C, H, E \rangle$

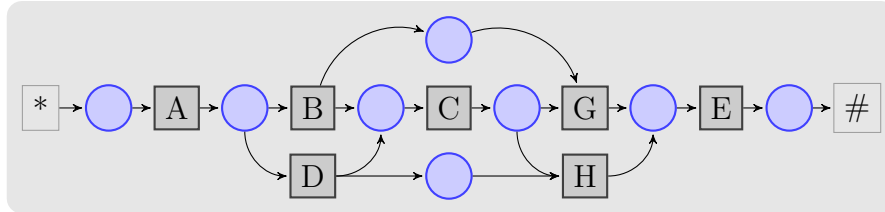


**Log 8 - Choice-relation:** Long-term choice relationship. This log is similar to the last one, but a more correct models bounds the decisions made in different points in the flow of the process.

This means, there are two new places added to the last model to force a trace passing through B to pass through G later and, similarly, another place bounds D to H.

Log 8:

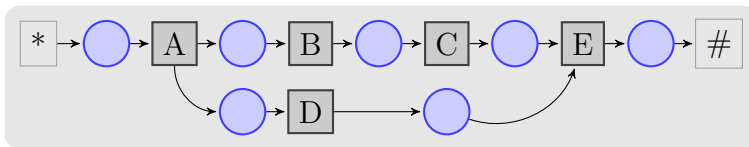
- $\langle A, B, C, G, E \rangle$ ,
- $\langle A, D, C, H, E \rangle$



**Log 9 - Asym. AND:** Asymmetric parallelism. Just like the previous log, this one tests the ability of perceiving parallelism with slight scaling. This example introduces the difficulty of perceiving the consecutive relationship between activities B and C.

Log 9:

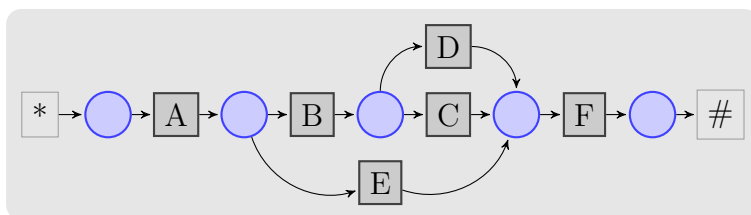
- $\langle A, B, C, D, E \rangle$ ,
- $\langle A, B, D, C, E \rangle$ ,
- $\langle A, D, B, C, E \rangle$



**Log 10 - Nested 2OR:** Nested decisions. This log increases the difficulty of previous decisions. It also has an interesting place with three entering arcs.

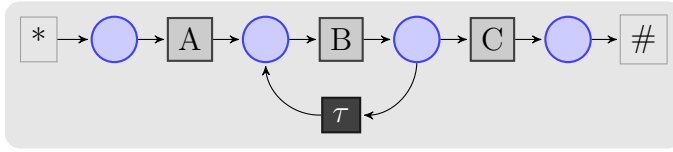
Log 10:

- $\langle A, B, C, F \rangle$ ,
- $\langle A, B, D, F \rangle$ ,
- $\langle A, E, F \rangle$



**Log 11 - Oblig. Loop:** Loop with at least one passage. This log represents a process with loop that must be executed at least one. It is different from the variant previously presented, where there was an option not execute the activity B.

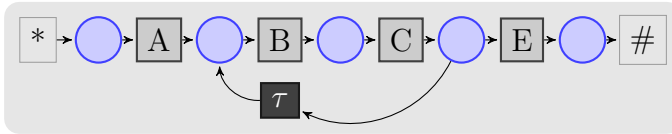
Log 11:



- $\langle A, B, C \rangle$ ,
- $\langle A, B, B, C \rangle$ ,
- $\langle A, B, B, B, C \rangle$

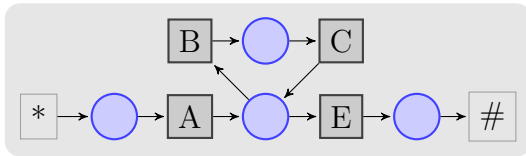
**Log 12 - Oblig. Loop large;** Larger loop with at least one passage. This log scales the concept introduced in the last one.

Log 12:



- $\langle A, B, C, E \rangle$ ,
- $\langle A, B, C, B, C, E \rangle$

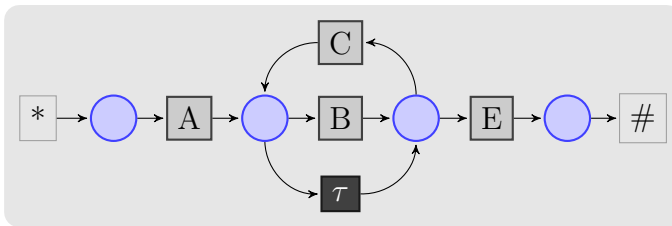
**Log 13 - Optional loop:** Larger optional loop. This log allows flow to ignore the loop.



Log 13:

- $\langle A, E \rangle$ ,
- $\langle A, B, C, E \rangle$ ,
- $\langle A, B, C, B, C, E \rangle$

**Log 14 - Nested loop:** Nested loops or loop with skip. This log can be modeled with many other representations. We present below a simple alternative, which can lead to logs not stated in the log.



Log 14:

- $\langle A, E \rangle$ ,
- $\langle A, B, E \rangle$ ,
- $\langle A, B, C, B, E \rangle$

The primary goal of this section is to test the new formulation and some minor variations of it. Demonstrating all combinations of variants would be impossible. Therefore, we established a base case and some variants to better understand the final effect of the formulation onto the process Petri nets discovered.

We keep some aspects of the mathematical model in all variants. These aspects are:

- The objective function is always a minimization of a weighted sum, although different variants use different weights.
- All variants have the constraints that define variables  $a_{ij}^k$  (consecutive connection between activities  $i$  and  $j$  in place  $k$ ) and variables  $w_{ij}$  (consecutive connection between activities  $i$  and  $j$  in at least one place).
- We suppose a fixed number  $k$  of places to all variants. They are modeled to return with zero entering and leaving arcs when they are not used.
- All Variants define constraints on  $X$  and  $Y$  based on the log's prefix closed language (the set of all traces and respective prefixes)

**Base case - Variant zero** Corresponds to the GILP formulation above with  $W_1 = 5$  and  $W_2 = -1$ . Additionally, we consider: no trace suffixes constraints, parallelism prohibition using  $w_{ij}$  as above, fixing of start and end, and token handling symmetry

**Variant 1 - OF:W** objective function keeps weight 5 in  $w_{ij}$  but gives zero weight for  $x_i^k$  and  $y_i^k$  variables;

**Variant 2 - OF:XY** objective function keeps weight -1 for  $x_i^k$  and  $y_i^k$  variables but gives zero weight for  $w_{ij}$  variables;

**Variant 3 - OF:XYZ** objective function with zero weight for  $w_{ij}$  variables, weight of 5 for the  $z_{ij}^k$  variables and keeps weight -1 for  $x_i^k$  and  $y_i^k$  variables;

**Variant 4 - Suffixes** include suffixes constraints;

**Variant 5 - XY parallel.** constraining parallelism using  $x_i^k$  and  $y_i^k$  variables instead of the  $w_{ij}$  variables;

**Variant 6 - w/o W fix.** exclude Start and Finish fixing on the  $w_{ij}$  variables;

**Variant 7 - token sym.** forcing token symmetry;

**Variant 8 - w/o flow forcing** exclude constraints (4-7).

		Base Case	Variant 1	Variant 2	Variant 3	Variant 4	Variant 5	Variant 6	Variant 7	Variant 8
		OF: W	OF: XY	OF: XYZ	Suffixes	XY parallel	w/o W fix	Token sym	w/o flow-forcing	
Log 1	Linear	ok	ok	ok	ok	ok	ok	ok	ok	missing
Log 2	Skip/loop	wrong	wrong	ok	wrong	wrong	wrong	wrong	wrong/dup.	missing
Log 3	OR-open	ok	ok/dup	ok	ok	ok	ok	ok	ok	missing
Log 4	asym. OR-open	ok	ok	ok	ok	ok	ok	ok	ok	missing
Log 5	AND-open/close	wrong	wrong	wrong	wrong	wrong	wrong	wrong	ok	missing
Log 6	OR-open/close	ok	ok	ok	ok	ok	ok	ok	ok	missing
Log 7	2OR-open/close	ok	wrong	ok	ok	ok	ok	ok	ok	missing
Log 8	Choice-relation	wrong*	wrong	wrong	wrong*	wrong*	wrong*	wrong*	wrong*	missing
Log 9	Assym AND	wrong	wrong	wrong	wrong	wrong	wrong	wrong	ok	missing
Log 10	Nested 2OR	ok	wrong	wrong	ok	ok	ok	ok	ok	missing
Log 11	Oblig. loop	wrong	wrong	wrong	wrong	wrong	wrong	wrong	wrong/dup.	missing
Log 12	Oblig. loop large	infeasible	infeasible	infeasible	infeasible	infeasible	infeasible	wrong	infeasible	missing
Log 13	Optional loop	ok	ok	ok	ok	ok	ok	ok	ok	missing
Log 14	Nested loop	wrong	wrong	wrong	wrong	wrong	wrong	wrong	wrong	missing

Figure 4.2: Results table. It compares the outcomes of each of the 14 small synthetic instances to 9 variant setups. It is possible to deduce that the auxiliary ideas in the formulation do not help much and that the intrinsic difficulty lies within the instance itself.

#### 4.4 Results

Abbreviations used in the table above:

- **ok** - places found by the algorithm were expected;
- **wrong** - the algorithm found at least one wrong place;
- **dup** - some places found were duplicate;
- **missing** - the algorithm found some correct places but not all;
- **infeasible** - integer program generated does not have a feasible solution;
- **wrong\*** - specifically in Log 8, there were two places that all algorithms have difficulties in finding. These tests did not find those special places.

Variant 8 showed that turning off the graph connectivity (flow forcing) constraints makes the model find only the start and finish places. Therefore, the correctness of all models relies on this set of constraints. Unfortunately, this may be a problem for large instances, because the size of this set of constraints grows exponentially relation to the alphabet.

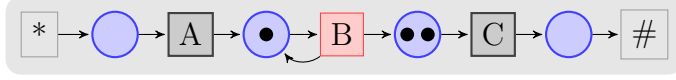
Additionally, the algorithms and its variants found wrong places in logs with AND-gates (parallelism). In Logs 5 and 9, the algorithm could find the places that form an AND-gate opening but closed the parallel streams with an OR-gate.

Logs which we predicted the need for a  $\tau$  transition were all wrong. The formulation still does not provide a way to model  $\tau$ . However, the Petri nets obtained keep their ability to replay in exchange for its soundness.

The example below shows the state in the Petri net after the second firing of transition B in log 11. Although this workflow creates tokens indefinitely in

the place before C, violating the proper completion property of the Petri net, the Petri net still can replay the log but is not sound.

Log 11:



- $\langle A, B, C \rangle$ ,
- $\langle A, B, B, C \rangle$ ,
- $\langle A, B, B, B, C \rangle$

Log 2 has a double nature. It can be either modeled with a simple cycle or using tau. Variant 2 was able to capture the intricate place that handles the sequence and the loop.

Some cycles of more significant size can make the model infeasible. It happened to the experiments in log 12. Nevertheless, the optimization model was feasible in variants 6 and 8 which are have significantly less constrained than the others. However, log 13 was captured by all Variants correctly.

		Base Case	Variant 1	Variant 2	Variant 3	Variant 4	Variant 5	Variant 6	Variant 7	Variant 8
		OF: W	OF: XY	OF: XYZ	Suffixes	XY parallel	w/o W fix	Token sym	w/o flow-forcing	
Log 1	Linear	0,89	0,14	0,21	0,20	0,16	0,17	0,25	0,13	0,13
Log 2	Skip/loop	0,25	0,16	0,15	0,26	0,22	0,27	0,91	0,28	0,09
Log 3	OR-open	0,13	0,11	0,14	0,13	0,11	0,12	0,70	0,11	0,12
Log 4	asym. OR-open	0,15	0,13	0,25	0,14	0,15	0,18	4,54	0,33	0,25
Log 5	AND-open/close	1,24	0,33	0,39	0,56	0,95	0,69	2,00	0,44	0,13
Log 6	OR-open/close	0,18	0,18	0,18	0,18	0,16	0,18	2,12	0,35	0,21
Log 7	2OR-open/close	1,14	0,74	2,30	1,11	1,19	1,39	972,39	0,57	0,32
Log 8	Choice-relation	1,92	0,92	3,43	0,93	0,98	1,17	425,04	0,79	0,14
Log 9	Assym AND	0,20	0,17	0,35	0,53	0,78	0,46	4,61	0,64	0,19
Log 10	Nested 2OR	1,24	1,68	1,13	0,96	1,69	1,58	272,32	1,55	0,24
Log 11	Oblig. loop	0,19	0,15	0,22	0,24	0,20	0,19	0,23	0,21	0,12
Log 12	Oblig. loop large	0,09	0,09	0,09	0,09	0,09	0,18	1,92	0,10	0,21
Log 13	Optional loop	0,62	0,52	0,55	0,70	0,73	0,66	10,31	0,81	0,24
Log 14	Nested loop	0,57	0,53	0,80	0,83	0,79	0,63	9,52	1,12	0,24

Figure 4.3: Execution times of the experiments in seconds. Across variants, the experiment without fixing initial variables took much longer. The degree of freedom in the  $w_{ij}$  network increases severely the execution times.

The table above shows the execution time in seconds of the optimizers. In terms of computing times, the only model that did poorly was the one without fixing the start and end activities. The Logs that had a significantly worse time-efficiency were Logs 7 and 8. Interestingly, those have a greater variety of activities, which translate to more variables in the optimization model. (VanDerAalst2011)

The global ILP formulation introduces the idea of using classical solvers for mixed integer linear programming to select places of the Petri net. Other approaches tested in the literature include greedy heuristic strategies, and solutions filtering after a round of solutions (places) enumerations.

Using integer linear programming also for place selection has the advantage of using the flexibility of the mixed integer linear programming modeling language to introduce cohesion or other desired aspects for selecting places.

As discussed in the last section, the global formulation in its compact version was only capable of handling small problems. However, the formulation has two exploitable characteristics from an integer programming decomposition standpoint. Firstly, the large number of constraints which enforce flow are not all needed for all of them to be feasible. Therefore, we can establish an oracle for selecting testing constraints and introducing them lazily into the solution.

For larger problem instances, it is possible to observe a combinatorial explosion of feasible places. Just a few of these are relevant to forming the final Petri net though. Therefore, if we were to replicate the subproblem a large number of times, it would build-up to a problem size of untractable proportions. We also can apply the same principle as to the constraints and use a variation of the classic formulation as place/variable oracle and introduce them lazily to the main program.

We then divide the algorithm into three modules: a Master, a place pricer and a flow-enforcing separator. The Master searches for a viable integer solution guaranteeing cohesion between places and calls the other modules whenever necessary.

A recursive procedure controls the branching of different fixations made to the relaxed master for each binary  $w_{ij}$ . Whenever the solution from a specialized linear solver for the relaxed master is within  $\epsilon$  of every binary variable being an integer, the branching procedure saves a new Petri net solution and calls the procedure that evaluates it.

There is a linear solver which verifies the reduced cost to call the pricing subproblem and the minimum cut in the  $w_{ij}$  network graph. It calls a standard solver to solve the relaxed master. Then verifies both: (1) if there are negative

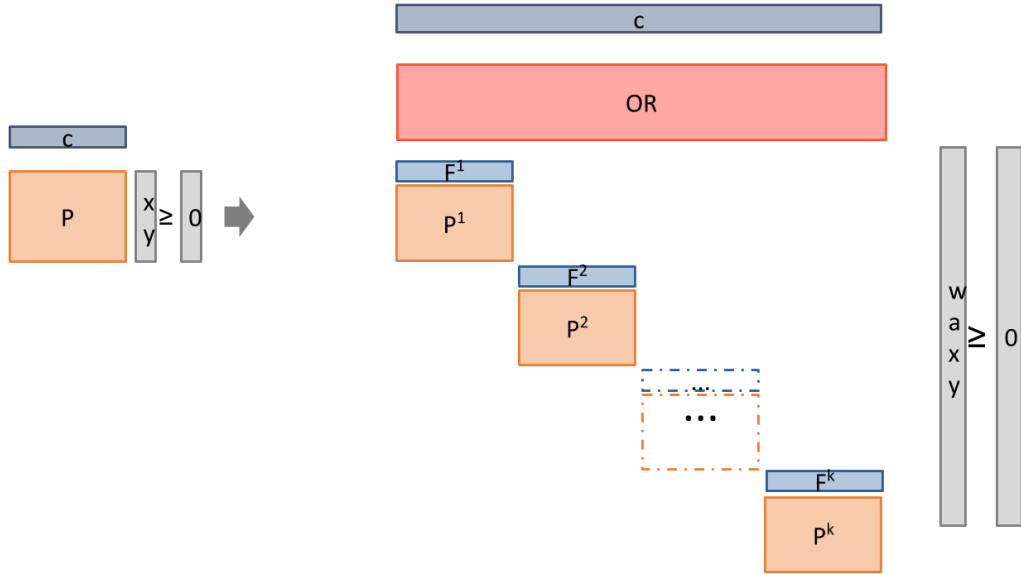


Figure 5.1: Graphical representation of the Global Model non-zeros of the formulation. The main idea was to replicate the classical formulation  $k$  times and to enforce cohesion using general constraints on a direct network linking events to events. A block-structure like this with linking constraints suggests the opportunity to apply the Dantzig Wolfe decomposition method for efficiency gain.

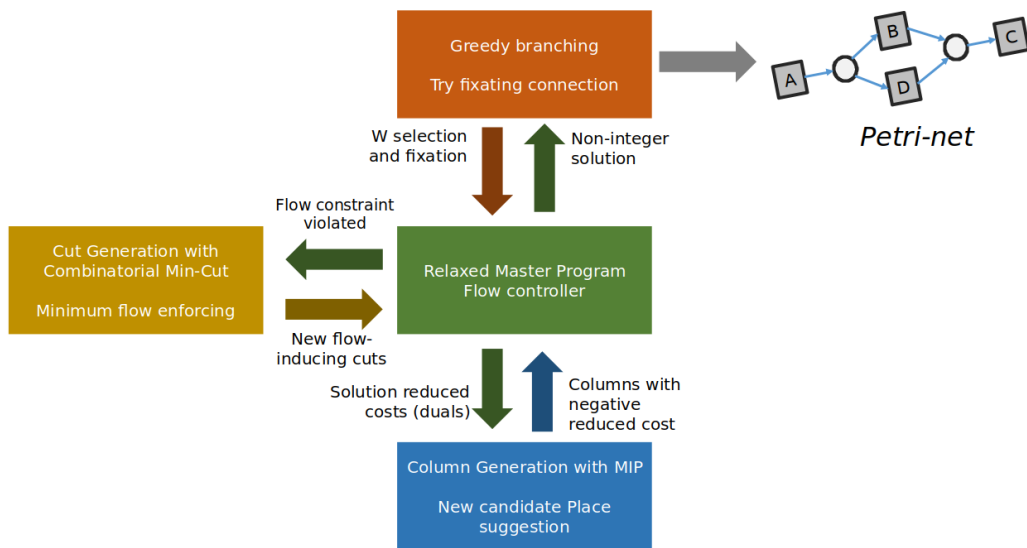


Figure 5.2: Branch-cut-and-price algorithm schematics. A recursive procedure controls the branching of different fixations made to the relaxed master. There is a linear solver which verifies the reduced cost to call the pricing subproblem and the minimum cut in the  $w_{ij}$  network graph.

reduced costs through a pricing problem indicating that there are columns to be generated (new place candidates) and (2) whether the min-cut algorithm has objective function below one, which would indicate that there is at least one



flow-enforcing constraint that needs to be added to the problem. This linear solver loops until the reduced cost are within an approximation parameter  $\epsilon$ , and the min-cut procedure indicates that there is no new cut to be added.

The Place oracle and flow oracle will be presented separately in the following sections.

## 5.1

### Place oracle (Column Generation)

The global integer formulation presented in the previous section replicates the original subproblem  $k$  times in order to produce up to  $k$  candidate places to form a Petri net. Although, in some cases, the event log is so complex that the model will only be feasible using a great amount of replicated programs into a single program creating an intractable amount of integer variables for any modern solver to work efficiently. As a proof of concept, we wanted to test how the GILP model behaved in toy problems that expressed minimally the known problems that all process discovery algorithms allegedly face.

For larger instances, we propose a Dantzig-Wolfe decomposition of the GILP model presented in the last section. This decomposition is useful for problems with many called separable constraints and a large number of variables.

Suppose a subset of the problem constraints does not constrain all variables. An even stronger supposition would be whole blocks of constraints just constraint their own subset of variables, and no single constraint acts upon variables from different blocks. In this situation, one could break the optimization problem into subproblems that use each a subset of variables for each block separately and then sum the objective functions.

A problem arises when there are constraints which connect variables which belong to different blocks. We call them complicating constraints. The value of the decision variables to each subproblem solved separately would not necessarily be feasible by the complicating constraint (Conejo2006). However, it is possible to use the phenomenon of complementary slackness to solve the problem efficiently. Strong duality theory (Chvátal1983) describes the phenomenon of two dual programs having an optimal solution of equal value and for each pair of primal variable and dual constraint or each pair of dual variable and primal constraint will have either value zero and non zero slackness or non zero value and zero slackness.

Usually many decision variables in a standard linear programming problem have their optimal value equals to zero. The geometric interpretation of linear programming gives us a glimpse of it. If we imagine that we have an

objective function as a direction to go in a convex constrained space, the final destination of the walk will be a constraining corner. The boundaries forming this optimal corner will have slackness zero whereas all others will have slackness different than zero.

In the space of the dual program, dual variables associated with constraints that are tight (zero slackness) will have a nonzero value. Roughly, the dual variables can be interpreted as partial derivatives of the objective function or the expected gain of relaxing the constraint by one unit.

If we could guess which of the variables have some probability of having a non-trivial (above zero) optimal value, we could solve the problem without even considering trivial variables. When there is a large set of block separable constraints, we could try to identify which of the variables have negative associated reduced cost, which means that they could reduce our minimization objective function. Then it would be possible to reinsert them into a master program containing the original objective function and the original complicating constraints.

With the dual variable's value obtained solving this master program can be used to generate a function of the reduced cost contribution using the variables in each block (Ford1958). Therefore, the block subprograms can use this function of the reduced cost to find which variables of each block would have minimal reduced cost and therefore the chance of actively contributing to reducing the value of the Objective function in the master program.

First, the GILP compact formulation is comprised of the replication of many subprograms with the original formulation and the Fortet's transformation of the places into bicliques. Therefore, at the core of our algorithm lies solving a master program comprised of GILP's objective function and complicating constraints and a subproblem comprised of the original trace constraints, the Fortet transformation and a surrogate objective function which minimizes the master's reduced cost.

Practically, we think of the program as two separate objects. The master program tries to cover the event-to-event network described by  $W$  variables using the bicliques which now will be generated by the subproblem. The dual variables form the reduced cost that informs the subproblem where to generate more places. Moreover, the subproblem uses the dual information to identify all possible places that may be selected by the master program to form the optimal Petri net, then the candidate places identified with negative reduced cost are reinserted in the master program encoded as bicliques described with the appropriate  $A$  variables. We solve both of them until the minimum reduced cost is within an  $\epsilon$  approximation of zero.

Formally, the Dantzig-Wolfe procedure prices variables using a pricing subproblem (Dantzig1960). This place pricer amounts to solving one place at a time by using the classic formulation similar to in the base case described by (Bergenthum2007), (VanDerWerf2008), and (VanDerWerf2009).

We present then the decomposition of the GILP formulation into a master relaxed problem and a subproblem that finds places minimizing the masters reduced cost as an objective function.

### Master Formulation

$$\min \quad W_1 \sum_{(i,j) \in A \times A} w_{ij} + W_2 \sum_{p \in \mathcal{P}} \lambda_p + W_3 \sum_{p \in \mathcal{P}} d_p \lambda_p \quad (5-1)$$

$$\sum_{i \in D} \sum_{j \in A-D} w_{ij} \geq 1 \quad \forall D \subset A, * \in D, \# \notin D \quad (5-2)$$

$$(dual : \pi_{ij}) \quad \sum_{p \in \mathcal{P}} (a_{ij}^p \lambda_p) - w_{ij} \geq 0 \quad \forall (i, j) \in A \times A \quad (5-3)$$

$$(dual : \bar{\pi}_{ij}) \quad M.w_{ij} - \sum_{p \in \mathcal{P}} (a_{ij}^p \lambda_p) \geq 0 \quad \forall (i, j) \in A \times A \quad (5-4)$$

$$(dual : \gamma) \quad \sum_{p \in \mathcal{P}} \lambda_p \leq k \quad (5-5)$$

$$(dual : \beta) \quad \sum_{p \in \mathcal{P}} b_p \lambda_p = 0 \quad (5-6)$$

We leave in the master formulation the complicating constraints such as the ones that connect each  $w_{ij}$  to associated  $a_{ij}^p$ . The objective function 5-1 is represented in a generic form, using the constants  $W_1, W_2, W_3$  as interchangeable costs for future tests. In preliminary tests of the GILP model we discovered that pricing  $w$  variables and  $x, y$  worked well. The coefficients used in the objective function are still an open research question for further development. By this work, we have tested in the experimental setup using  $W_1 = 50, W_2 = 0, W_3 = 10$ . So, we receive from the subproblem a coefficient  $d_p$  that has the information of how many arcs  $(\sum x + \sum y)$  a place indexed with  $p$  has. We still do not understand how it would affect to define  $W_2 \neq 0$ .

Constraints of the family depicted in 5-2 are too numerous and we will also add them lazily using a cut generating procedure based on solving a minimum cut problem using classic algorithm for maximum flow. Equations 5-3 and 5-4 control the connection between the selection of places using decision variables  $\lambda_p$  and the generated column  $a_{ij}^p$  and variables  $w_{ij}$ . For example, if place  $p_5$  is selected,  $\lambda_{p_5}$  should be equal to one pushing to one as well all  $w_{ij}$  for all  $ij$  that place's  $p_5$  biclique (denoted as  $a_{ij}^{p_5} = 1$ ). So,  $w_{ij}$  will be pushed to one whenever there is at least one selected biclique  $p$  that has the edge  $ij$ .

Constraint 5-5 is not used, we determine an arbitrary big  $K$  so that it will not interfere with results. It sets a bound on how many places can be selected

for a Petri net. Future work might use it. Constraint 5-6 would enforce the sum of  $x$  and  $y$  to be equal in an attempt of mitigating overproduction of tokens. We also do not use it and, in practice, we put a slack variable there as an indicator of how many more arcs there are being produced than consumed.

### Acceptable numeric gap

$$\text{if } \exists p \quad | \quad \bar{C}_p < 0 \quad (\bar{C}_p < \epsilon) \quad \epsilon \approx 10^{-3} \quad (5-7)$$

The master formulation will run using relaxed  $w_{ij}$  and  $\lambda_p$ . The fixation of the values of  $w_{ij}$  to zero is controlled by the outer loop of branching. This outer branching procedure calls a relaxed solver, that iteratively solves the relaxed master and calls the pricing problem below. The pricing problem has the goal of showing that the master is solved using all variables that have the potential of reducing further the masters objective.

Whenever there may exist a variable that may reduce the objective function, the pricing problem should identify it and add to the master problem (as a new column indexed as  $p$ ). These variables show themselves by presenting negative reduced cost calculating using the master's duals. In practice, we run the master and the pricing problems multiple times and add minimal reduced cost columns until the minimum reduced cost is within an epsilon approximation distance of zero. Only then, we can affirm that there are not any variables that could further improve the objective.

### Subproblem Formulation

$$\min \bar{C} = W_2 - \gamma + -\beta b + W_3 d + \sum_{(i,j) \in A \times A} [(\bar{\pi}_{ij} - \underline{\pi}_{ij}) a_{ij}] \quad (5-8)$$

$$x_* - y_q + \sum_{j \in \sigma_1} (x_j - y_j) \geq 0 \quad \forall \sigma_1, q \mid \dots \text{see description} \quad (5-9)$$

$$a_{ij} \leq x_i \quad \forall (i, j) \in A \times A \quad (5-10)$$

$$a_{ij} \leq y_j \quad \forall (i, j) \in A \times A \quad (5-11)$$

$$a_{ij} \geq x_i + y_j - 1 \quad \forall (i, j) \in A \times A \quad (5-12)$$

$$d = \sum_{i \in A} (x_i + y_i) \quad (5-13)$$

$$b = \sum_{i \in A} (x_i - y_i) \quad (5-14)$$

The remaining constraint of the problems that manifest themselves as blocks in the GILP formulation presented on section 4 are optimized as the pricing problem using the reduced cost as objective function 5-8. Constraint 5-

9 depicts the classic trace constraints where this work all began. For simplicity purposes, we transferred the indexing of this family of constraints to here:  $\forall \sigma_1, q \mid \exists \sigma \in S \text{ and } \sigma_2, \sigma = *. \sigma_1. q, . \sigma_2$ . This means that for all prefixes  $\sigma_1$  and last activity  $q$  of this prefix  $\sigma_1$  to the observed  $\sigma \in$  event log  $S$ .

Constraints depicted in 5-10, 5-11 and 5-12 are the Fortet linearized multiplication that turns the variable  $a_{ij}$  as a logical AND whenever  $x_i$  and  $y_j$  are set to one. As we presented before, this turns the place generated as a biclique from event to event, facilitating the selection and covering as we add  $a_{ij}$  to the master formulation as new columns of coefficients indexed  $p$ .

Constraints 5-13 and 5-14 are just there to inform the master the sum of arcs used by the place and balance between arcs leaving and entering a given place.

## 5.2

### Alternative place oracles

Many process discovery algorithms such as the alpha algorithm, heuristics miner and the ILP miner itself rely on the strategy of finding places that form a Petri net. They work from an inside-out perspective. Other algorithms, such as the inductive miner, rely on subdividing the event log into event sublogs recursively, analogously working from an outside-in perspective.

The Dantzig-Wolfe decomposition of the GILP formulation treats the Bergenthum formulation and the Fortet/McCormick translation as a place oracle. However, it is possible using other place oracles alongside the proposed in the last subsection. Given that several algorithms rely on finding candidate places, we could use them as a heuristic to generate places for the master program to use.

The only risk with this approach is generating places that are not feasible in with the ILP formulation and, thus, reducing the overall quality of the process because it has at freedom to choose from low-quality places to form the Petri net. The implementation is as follows: at the beginning of the execution of the algorithm, we can run any other process mining algorithm that produces places, fixate the variables of each of these places into a copy of the column generation procedure and optimize. With the fixation, we expect that some of the places are going to be feasible in the ILP formulation and, therefore, can be added as valid columns in the relaxed master procedure. These copies are used as a place filter.

Another way is to use the outside-in algorithmic approaches to process discovery. As seen in the previous chapter, the original formulation obligates a Parikh vector over prefixes multiplying binary  $X$  and  $Y$  variables to be



log in some trace the event  $i$  is directly followed by event  $j$ . The two of the four examples of operators that Leemans proposes to subdivide the event log are the *sequence* operator and the *exclusive choice* operator.

The *sequence* operator has the characteristic of identifying in the directly follows graph groups of nodes in which when going out from the group through an edge there is no path back. The *exclusive choice* operator identifies groups of nodes in which when we are in one is not possible to arrive in the other and vice versa. All of these cases can be efficiently found using one execution of the Kosaraju's two-pass Depth First Search algorithm for identifying strongly connected components in  $\mathcal{O}(n^2)$  being  $n$  the size of the alphabet.

An event log without many large inversions (which usually characterizes parallelism) or many repetitions of the same event in the same case (which carry represent cycles) could be broken down into small strongly connected components and consequently small subpartitions of the alphabet.

Then for each subalphabet related to each strongly connected components, we built a smaller pricing subproblem using just the events in this subalphabet. Therefore, we hope to generate valid columns much more efficiently than using just the original pricing subproblem.

We have shown in section 3 that if the places are feasible for the filtered subproblem, they are feasible for the original formulation. Furthermore, when building these smaller programs, we can plug in current duals because the variables do not change, because we suppose that the filtered log approach is entirely equivalent to fixating variables not associated with the sub-alphabet to zero.

### 5.3

#### Column generation stabilization

Just as an implementation note, the introduction of heuristic columns at an early point of an algorithm experienced numerical instability in early undocumented tests. Although some cases experienced a significant reduction of execution time due to the introduction of these columns, other cases have experienced columns which have reduced cost near the  $\epsilon$  constant used for numerical approximations.

The beginning of the execution of the algorithm, the places introduced to the model have huge calculated reduced cost and are composed of almost all arcs. We conjecture if early executions of the algorithm used these places and their associated columns introduced to the primary model as responsible for constraining the dual of the master relaxed problem. The introduction of the heuristic columns at the beginning of the execution increased the perceived

instability and execution times.

The two approaches used for mitigating this problem were: delaying the introduction of the heuristic columns and the use of weighted duals strategy for the pricing problem as described by Wentges in (Wentges1997). The first strategy yielded immediate results. We have set up a global counter for columns generated by the full pricing problem, and when a certain threshold is achieved, we add all columns produced by the alpha algorithm which are feasible by the Bergenthum's formulation and we allow the strongly connected components filtered pricing problem to run before the original pricing problem. This way, we believe that the dual of the master has enough constraints associated with the columns generated to stabilize.

The other strategy is to interactively run the pricing problem in many stabilization rounds dissolving the aggressiveness of the current dual by weighting it to a fixed feasible dual. In our problem, we know by the formulation that zero values are always feasible for the dual variables  $\beta, \gamma, \bar{\pi}_{ij}, \underline{\pi}_{ij}$ .

Therefore, we choose a weight of  $\alpha$  (we settled for 0.7) and build the pricing objective function using a weighted expression of the duals with a feasible one.

$$\beta_{\text{weighted}} \leftarrow \alpha\beta_{\text{current}} + (1 - \alpha)\beta_{\text{feasible}} \quad (5-15)$$

$$\gamma_{\text{weighted}} \leftarrow \alpha\gamma_{\text{current}} + (1 - \alpha)\gamma_{\text{feasible}} \quad (5-16)$$

$$\bar{\pi}_{ij,\text{weighted}} \leftarrow \alpha\bar{\pi}_{ij,\text{current}} + (1 - \alpha)\bar{\pi}_{ij,\text{feasible}} \quad (5-17)$$

$$\underline{\pi}_{ij,\text{weighted}} \leftarrow \alpha\underline{\pi}_{ij,\text{current}} + (1 - \alpha)\underline{\pi}_{ij,\text{feasible}} \quad (5-18)$$

The column generation procedure runs until the reduced cost is zero. This stabilization scheme introduces an outer loop of stabilization rounds to this original loop. For a predetermined number of rounds (we use 2 or 3 due to low chance of instability) we run the pricing loop with the weighted duals for calculating the reduced cost.

Whenever the stabilization round is through, we update our feasible duals to be the values of the weighted duals in the previous round. So, from going to stabilization round  $r$  to round  $r + 1$  we update the feasible duals:

$$\beta_{\text{feasible}} \leftarrow \beta_{\text{weighted}} \quad (5-19)$$

$$\gamma_{\text{feasible}} \leftarrow \gamma_{\text{weighted}} \quad (5-20)$$

$$\bar{\pi}_{ij,\text{feasible}} \leftarrow \bar{\pi}_{ij,\text{weighted}} \quad (5-21)$$

$$\underline{\pi}_{ij,\text{feasible}} \leftarrow \underline{\pi}_{ij,\text{weighted}} \quad (5-22)$$



In this way, the feasible duals get closer to where the current duals stand, and, consequently, the newly weighted duals will be closer to the current.

## 5.4

### Cut generation

Besides the combinatorial number of feasible places, there is also an exponential number of possible constraints for enforcing flow through the global network described by the  $W$  variables. We want every event to have at least one incoming and one outgoing  $W$  edge to every transition of the Petri net. Thus we enforce the “flow” from the artificial silent initial activity  $*$  to all combinations of events to be at least one and also the flow from all combinations of events to the final silent activity  $\#$  to be at least one.

The main idea is that it is not necessary to add all the approximately  $2^n$  constraints to enforce this flow. Most of the times, these constraints tend to have an overlapped effect. Thus, to avoid packing more non-zeros than necessary into the relaxed master, we chose also to relax these constraints and create a subroutine that checks if all cuts in this graph have at least one unit of flow going in and going out.

Remembering, these relaxed constraints have the following form:

$$\sum_{(i \in D)} \sum_{(j \in A-D)} w_{ij} \geq 1 \quad \forall D \subset A, * \in D \quad (5-23)$$

In this equation, all subsets of events  $D$  contained in  $A$  which have the initial artificial event  $*$  must have the sum of outgoing arcs greater than 1. By outgoing arcs, we mean an arc that goes from an event in  $D$  to an event outside  $D$ .

Every time we solve the relaxed master, we can use the values that the solver found for the  $W$  variables, and build a flow network in which these values are the associated capacity of the edges. Then we use a Maximum flow/Minimum cut specialized algorithm as a black box in order to find which of these subsets of events  $D$  has the minimum cut value. The minimum cut value below one will indicate that there are some subsets  $D$  of activities which do not hold. Then, until the minimum cut value of the network has value one, this subroutine produces a round of cuts and adds them to the relaxed master and resolve the relaxed master.

## 5.5

### Branch and bound

The branching is performed in a simple manner of fixating  $W$  variables to one and then to zero. We first select the  $W$  variable closest to 0.5 and fixate

them to 1. Recursively, we solve the linear problem once more and choose among the remaining variables the closest to one for fixation.

The recursion path is analogous to performing a depth-first search on a tree with a branching factor of two. In each of the nodes, we call the procedure that solves the relaxed master. This procedure guarantees that the minimum cut of the solution is at least one (which mean the cutting problem is solved) and that the minimum reduced cost among candidate places is zero (pricing problem is solved) (Vanderbeck2005).

At a certain point into the deep dive of fixations, the algorithm obtains a first solution in which all  $W$  variables are either above  $1 - \epsilon$  or below  $0 + \epsilon$ . In this situation, we consider having mined an integer solution in  $W$ . We then save the objective function value, and the associated Petri net by looking at which of the  $\lambda_p$  associated with generated places are above 0.1. Almost always the  $\lambda_p$  variables are an integer as well as the  $W$ , and, in the rare cases in which there are fractional  $\lambda_p$  they are just two or three very similar places. We decided to include all fractional places in these sporadic cases.

With a feasible integer solution in hands, we can use its objective as an upper bound for pruning the recursion tree before going too deep to get integer solutions. As we present in the following subsections, the first integer solution is usually so good that the procedure only finds one to five better solutions.

## 5.6

### Experimental setup

The decomposed formulation yields an algorithm capable of handling complex instances. Therefore, we decided to experiment with its performance against one of the most difficult synthetic instances available: the Process Discovery Contest 2016 and 2017.

During the execution of the algorithm, every integer solution which has an evaluated objective function better than the last produces a new PNML (Petri net XML) file, and we measure their replay fitness. This measure theoretically tries to measure how much of the behavior does the Petri net can reproduce from the given log.

**Experiment indicators** Our primary objective is synthesizing expressive models, which means: readable by humans and re-enactable by machines. Our chosen model is the Petri net. Unfortunately, it is possible to obtain process models that have artificially good characteristics, but in practice do not encode process characteristics such as flow diversity, loops and kinds of decisions.

Therefore, to measure how good a model remains an open question in the field.

As stated in Chapter 2, Aalst in (VanDerAalst2016) proposes that an acceptable process model should then balance four aspects of the process discovery task: fitness, precision, simplicity, and generalization. Simplicity is defined as how complicated is a model. If the model has a graphical representation, a simple model would use a few connections and would have a streamlined morphology instead of entangled connections.

Simplicity is an aspect of difficult measure objectively. What we can guarantee to impose a minimum standard of simplicity is that our algorithm:

- produces models which use only one transition for each symbol of the alphabet;
- we do not make use of silent transitions in the middle of the model, besides from one at the beginning and one at the end;
- our objective function penalizes the use of the arcs, so we should expect the most economical arrangement of valid places which minimally connect the Petri net.

As to the definitions of Fitness and Precision, we will use metric proposed by recent work by Aalst, Carmona, Munhoz-Gama in (VanDerAalst2016), (Carmona2018), (Adriansyah2015).

**Token-Replay Fitness** There are some alternatives to measuring fitness, the simplest one is the percentage of the sequences that can be correctly replayed in the model. As we discussed in the compact global model, this formulation still has limitations. For instance, it does not make use of silent activities and, consequently, cannot reproduce some basic behaviors as we observed with the small artificial cases in the last section. Additionally, processes themselves can be regionally good or bad. Therefore, this measure can be deceptive by rejecting an imperfect process with good regions.

A more flexible way of measuring Petri nets is proposed in chapter 7.2 of Aalst more recent book on process mining (VanDerAalst2016). We conduct the replay of every sequence onto the log, and we track the behavior of tokens through the activations of activities of the log using four measures. That is, during the replay of all sequences in the log, we update four counting variables:

- **C** - for the number of tokens **consumed**;
- **P** - for the number of tokens **produced**;

- **M** - for the number of **missing** tokens at the time an activity should be able to fire;
- **R** - for the number of tokens **remaining** at the end of the replay of each sequence.

With these counters, we derive indicators which roughly measure two undesired effects. Firstly, if we divide  $M$  by  $C$ , we can measure how many of the necessity of token consumption is not supplied. Also, the division of  $R$  by  $P$  denotes the percentage of produced tokens which are left after the simulation of the sequences. The smaller these percentages are, the fitter is the model to the log.

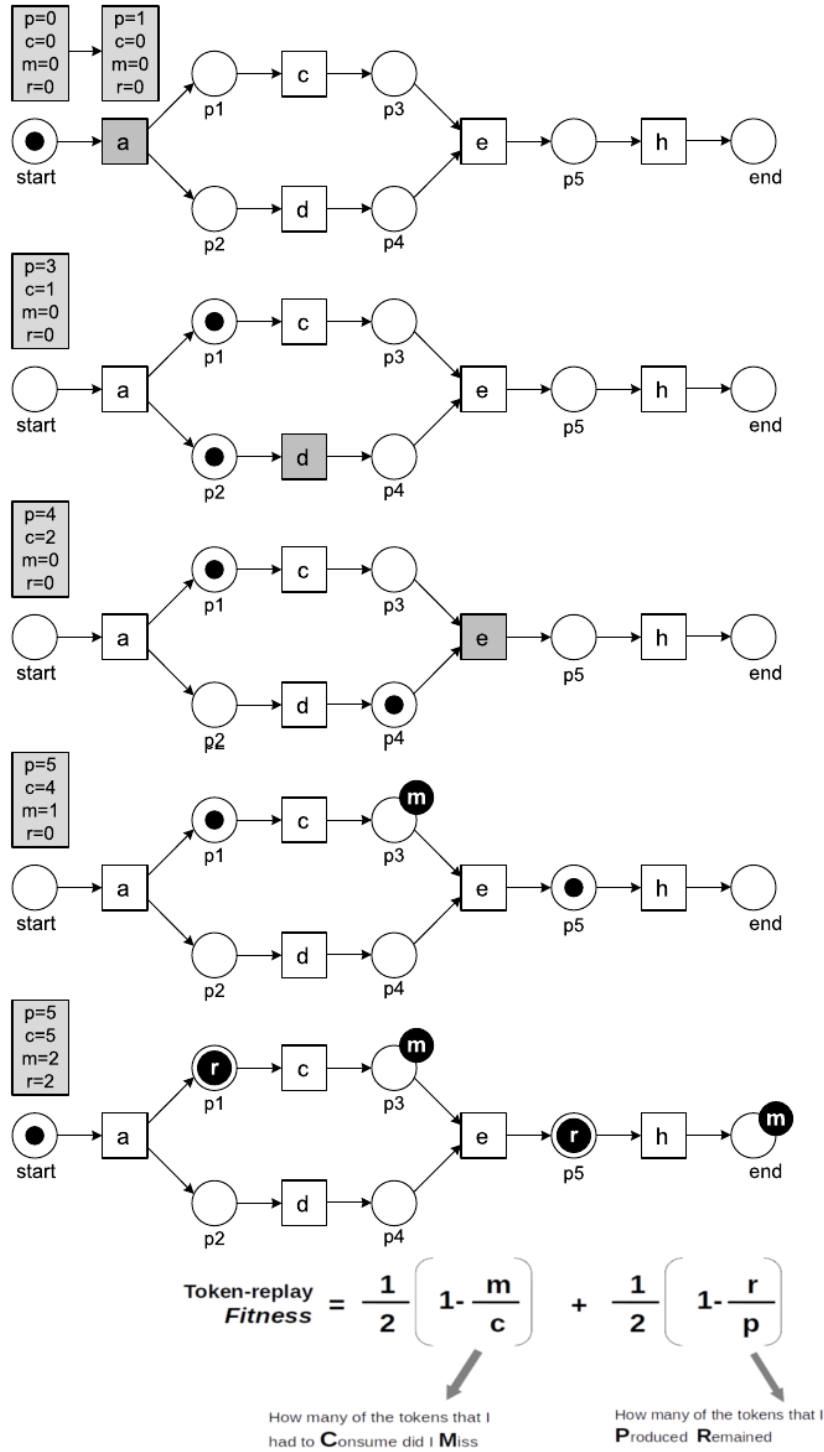


Figure 5.4: Example of the reproduction of a single log  $\langle a, b, d, e, g \rangle$  in a process model. During the execution, we maintain counters for  $C$ ,  $P$ ,  $M$  and  $R$ . Notice that the process model is flawed to a point where “ $b$ ” and “ $g$ ” are not even transitions in the process. Even with this characteristic we carry on forcibly activating the transitions which do belong to the model.

This alternative measure of fitness used throughout this dissertation combines them both. We then define this measure as a function of the replay procedure of the log  $\mathcal{L}$  over the Petri net  $P$  :

$$\text{replay-fitness}(\mathcal{L}, P) = \frac{1}{2}(1 - \frac{M}{C}) + \frac{1}{2}(1 - \frac{R}{P})$$

The fitness measure is the average of the complement of these indicators of undesired effects. We use the complement in order to make the score closer to 100% mean a perfect process and a 0% score the contrary.

**Leaving-Arcs Precision** For the precision, the recent book on conformance checking by Carmona ((Carmona2018)) demonstrates the difficulty of measuring precision and enumerates two significant approaches for measuring them. The approach used in this work is the leaving arcs approach. As explained in Munhoz-Gama's papers (Adriansyah2015) and (Munoz2010), an approximation is possible by exploring a tree of possible prefixes generated by a mined model iteratively checking the observed history. Therefore, the algorithm simulates sequences of activations possible by the Petri net building a tree until the Petri net reaches a final state or the algorithm prunes the search. The algorithm prunes the search whenever it recognizes a subsequence not present in the observed history. The generation of this simulation tree has internal nodes equal to prefixes observed in history and leaves that are connected to a final state of the Petri net or to leaving arcs, which mean, arcs that would lead to states not observed in history.

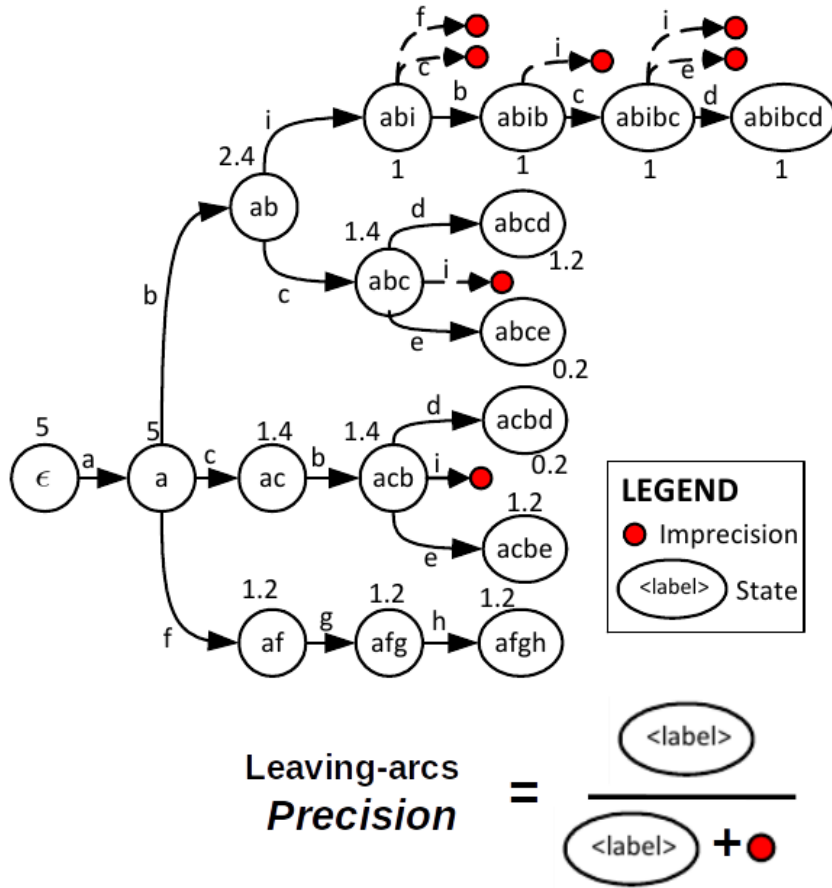


Figure 5.5: Example of a Petri net simulation tree adapted from Adriansyah’s paper on Measuring precision (Adriansyah2015). The internal nodes in white represent prefixes in the simulation of the Petri net which are present in the original event log. In red are the leaving arcs, which mean prefixes that can be generated by the Petri net but are not found in the event log. The approximation of the precision measure is the unweighted version, which is the simple division of internal nodes by the sum of internal nodes and leaving arcs of the tree.

They argue that a function over the internal nodes and the leaving arcs can be used as an approximation of the precision. They offer a variety of ways of measuring it and demonstrate in an experimental setup that all choices of functions achieve similar results. Therefore we have chosen the most straightforward unweighted version. Our precision measure is simply calculated as:

$$\text{precision}(\mathcal{L}, P) = \frac{n}{n+l}$$

Where  $n$  stands for valid prefixes found in the recursion and  $l$  stands for leaving arcs.

**Hardware and software used in inexperiment** We did all the implementation of the preprocessing, modeling and the control of the algorithm in Python 3.6. We used as an all-purpose LP solver for the relaxed master and MIP solver

for the subproblems the Gurobi suite 8.0.1 with an academic license and their respective API package for modeling problems in Python called “gurobipy”. Some standard packages were used in preprocessing as we such as Numpy and Pandas. Finally, we also used the NetworkX package for solving the maximum flow problem in the interface between the controller of the relaxed master and the cut oracle.

Also, we allowed the algorithm to process an instance for 24 hours. We interrupted the execution of the algorithm after this time limit.

The computer we ran the experiments is running the Kubuntu 16.04 OS. It has the Intel(R) Core(TM) i7-4790 CPU at 3.60GHz capacity and width 64 bits. Additionally, the memory of this computer has 31 GB of capacity.

**Instances** The problem instances we used for this problem are synthetic, created for the Process Discovery Contest proposed by the University of Eindhoven and available at (PDC2016) and (PDC2017). In each 2016 and 2017, they created ten process models that each reproduced different complicated aspects of the task of process discovery. Then they generated from these models some simulations in the form of sequences of events and provided only these sequences to the contestants.

The contestants were invited to submit process models in any language that captured the behavior of these sequences to be evaluated against traces generated by the process (not present in the data of training) and against traces not generated by the model (which the submitted models should not allow reproduction).

Each of these models contained at least two of the following characteristics:

- **Loops** Certain parts of the model can be repeated an arbitrary number of times.
- **Optional Activities** Certain activities are optional and can be skipped in certain runs of the process.
- **Inclusive Choices** Within the process, multiple sets of activities are optional, i.e., at least one set should be executed, but multiple sets of activities are also allowed. The difference with an exclusive choice resides on the fact that, in an exclusive choice, exactly one branch is activated; conversely, in an inclusive choice, more than one branch can be activated.
- **Recurrent activities** Activities can be executed in multiple non-subsequent points during runs of the process.



- **Long-term dependencies** A decision made at one point in the process can restrict the possibilities at subsequent decision points. For example, at the beginning of a process, a choice is made between an activity A and an activity B. When activity A is chosen, later during any run, an activity C cannot be executed; if activity B is chosen, activity C can still be executed. In the Petri net terminology, this corresponds to Petri nets with non-free-choice constructs.
- **Exclusive choices** can be characterized by balanced or unbalanced paths. If an exclusive-choice is characterized by being balanced, in any run of the process, each mutually exclusive set of activities has an equal probability of being chosen. If conversely, it is unbalanced, one set has a 90% probability of being chosen and the other sets, together, have 10%, with each of them having the same probability.
- **Incompleteness** Especially the instances of 2017 contained five processes with unfinished traces. In practice, this is a common feature because usually there are still cases in the middle of their processing when we extract a batch of data for mining.

The detailed comments of what characteristic each of the instances carried are in the results table in figure 5.6 we present next.

## 5.7 Results

The summary of the results is below. Each entry of this table shows details of each of the 20 instances and the performance of the final response of the algorithm.

case	case characteristic		XOR paths	alphabet	variants	largest trace					
	characteristic 1	characteristic 2					prec.	fit.	gap.	fini?	min.
2017.01	Optional activities	Long-term dependencies	Unbalanced	23	1000	18	12,21%	92,07%	0,79%	No	x
2017.02	Optional activities	Loops	Balanced	20	429	52	32,24%	95,56%	0,12%	Yes	6,43
2017.03	Optional activities	Inclusive choices	Unbalanced	21	477	18	27,92%	90,20%	1,02%	Yes	107,30
2017.04	Loops	Reoccurring activities	Unbalanced	20	744	60	15,41%	91,74%	1,01%	Yes	30,83
2017.05	Loops	Long-term dependencies	Balanced	23	997	61	19,29%	75,40%	0,95%	No	x
2017.06	Loops	Inclusive choices	Balanced	20	247	46	11,89%	96,42%	0,82%	Yes	2,70
2017.07	Inclusive Choices	Reoccurring activities	Balanced	12	307	11	18,16%	96,81%	1,45%	Yes	1,60
2017.08	Inclusive Choices	Long-term dependencies	Balanced	17	520	14	22,87%	89,46%	0,58%	Yes	85,35
2017.09	Reoccurring activities	Long-term dependencies	Unbalanced	22	563	25	66,67%	82,85%	0,71%	Yes	56,65
2017.10	Optional activities	Reoccurring activities	Unbalanced	14	711	17	42,31%	90,97%	0,83%	Yes	82,06
2016.01	Optional activities	Loops	Unbalanced	18	851	34	28,10%	89,06%	1,02%	Yes	649,09
2016.02	Optional activities	Inclusive choices	Balanced	20	907	16	9,30%	88,63%	2,14%	No	x
2016.03	Optional activities	Reoccurring activities	Unbalanced	22	590	21	8,33%	93,82%	0,20%	Yes	35,43
2016.04	Optional activities	Long-term dependencies	Unbalanced	22	179	21	29,39%	92,18%	0,61%	Yes	24,68
2016.05	Loops	Inclusive choices	Balanced	28	970	86	10,39%	93,25%	2,37%	No	x
2016.06	Loops	Reoccurring activities	Balanced	18	698	29	20,27%	94,00%	2,04%	Yes	174,23
2016.07	Loops	Long-term dependencies	Balanced	28	993	36	8,35%	91,92%	1,76%	No	x
2016.08	Inclusive Choices	Reoccurring activities	Balanced	18	411	12	9,02%	95,55%	0,82%	Yes	16,70
2016.09	Inclusive Choices	Long-term dependencies	Unbalanced	20	539	20	70,00%	92,31%	1,13%	Yes	10,68
2016.10	Reoccurring activities	Long-term dependencies	Unbalanced	20	960	14	11,80%	90,96%	0,96%	Yes	16,55

Figure 5.6: Table showing characteristics of each of the tested instances and the evaluation of the best solution found by our algorithm. We evaluated our algorithm by leaving arcs precision, token replay fitness, integrality gap and execution time of the algorithm.

Instances have at least two characteristics that usually adds difficulty to the process discovery task. The time of executing the algorithm is either very low or very high. Some instances have not finished the branching procedure after 24 hours running straight. Some instances after 24 hours have not even explored the first branch of depth ten entirely. So, after 24 for hours, the algorithm was still searching inside approximately 1/1000 of the search space.

The algorithm showed an overall consistency in producing high-fitness processes. Nevertheless, the precision is still a problem. Almost all instances had a remarkably low precision, denoting that the process models generated can produce much more behavior than what is necessary for the logs.

Each run of the algorithm produces multiple integer solution solutions, even though it is a small number. It seems to us that there are various possibilities of Petri nets with the same objective function using our proposed model because it does not matter how is the first path the branching explores, it always finds a good first solution quickly. The chart in figure 5.7 denotes the main milestones and the time it took to achieve that milestone.

Some instances in the chart in figure 5.7 have not finished but were still able to find the first solution. This behavior appears to be the only constant is that the algorithm can quickly find a valid Petri net and it takes a long time to finish. Also, the integrality gap being so low limits the maximum number of improvements possible, and also the expectation of finding new solutions with fewer arcs.

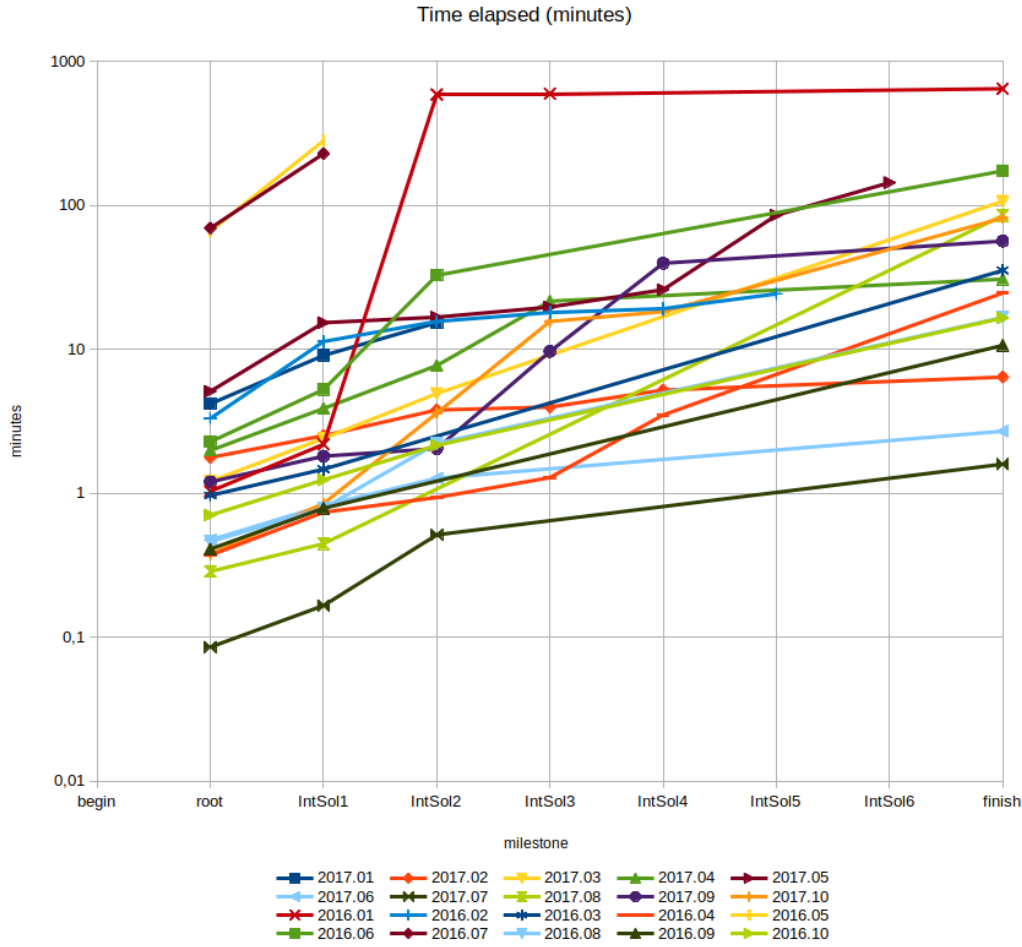


Figure 5.7: Elapsed time by optimization milestone. Before the algorithm starts branching, it finds a root solution. Then when the branching starts, it finds an integer solution (Petri net) and continues branching until it has tested all possibilities of fixating the root solution. Notice that the first solution the algorithm finds serves as bound for the branching procedure, and then the algorithm will only search for branches and solutions promising Petri nets with better objective function value.

Also, we want to explore the relationship between the advances in the integrality gap, the precision measure and fitness measure. To accomplish this, we build three scatter plots that show how the relation two-by-two of them. We also wanted to show all of the solutions found, not just the last; then it would be easier to conjecture the effects of improvements in the objective function and its reflection on the replay fitness and leaving arcs precision.

Firstly, we explore the relationship between the integrality gap and fitness. The integrality gap is how much the integral solution is away from the relaxed linear solution. So as long as the algorithm is running, it tries to close that gap. We observe in the chart of the figure 5.8, that as the algorithm progresses, the fitness measure seems to either stay at the same level or to

collapse at some random point for all instances.

Please notice the scale of each axis. They reveal that: the fitness measure always stays high for every solution we find and that every solution found has an integrality gap of less than 2.5%.

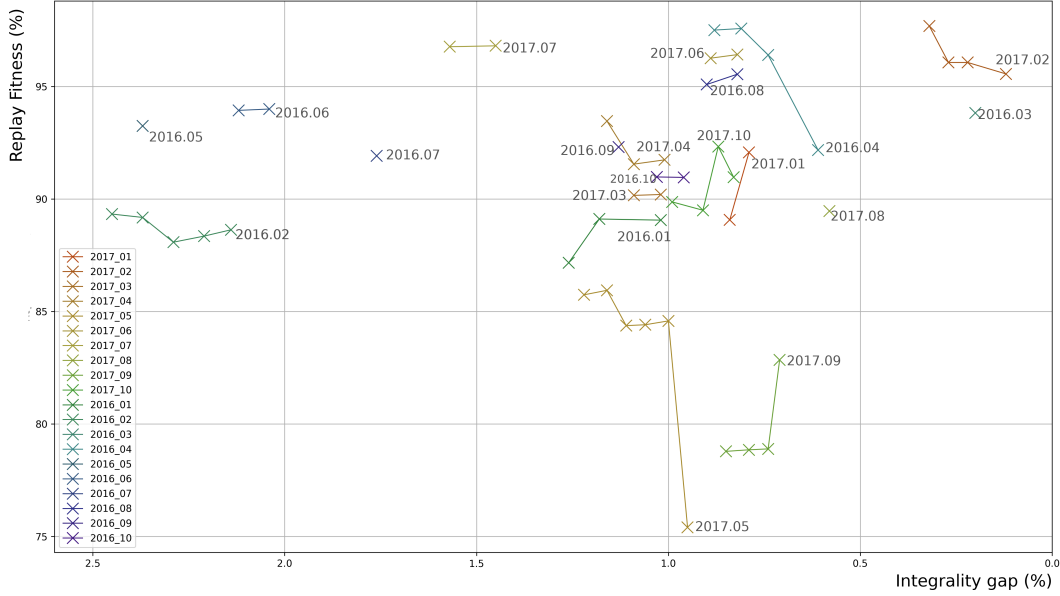


Figure 5.8: A scatter plot showing the relationship between the integrality gap and the fitness measure. The algorithm will find integer solutions with lower and lower objective function values relative to the root objective function. Therefore, we can visually interpret this chart as time flowing to the right as the algorithm progresses narrowing the integrality gap, notice that the right corner means gap equal to zero. The formulation for this problem seems to have a low standard integrality gap, and the objective functions seem disconnected to the fitness measure.

For precision, this effect is a little different as we show in figure 5.9. As the algorithm progresses closing the gap, the precision measure seems to either stay the same or to improve slightly in some instances. We conjecture whether the improvement in reducing the number of arcs also reduces the ability of the Petri net to produce tokens that would give way for leaving arcs. We explore these effects further in the experimentation of section 6, that explains the risk-prone method.

Either way, most instances also seem to have very low precision.



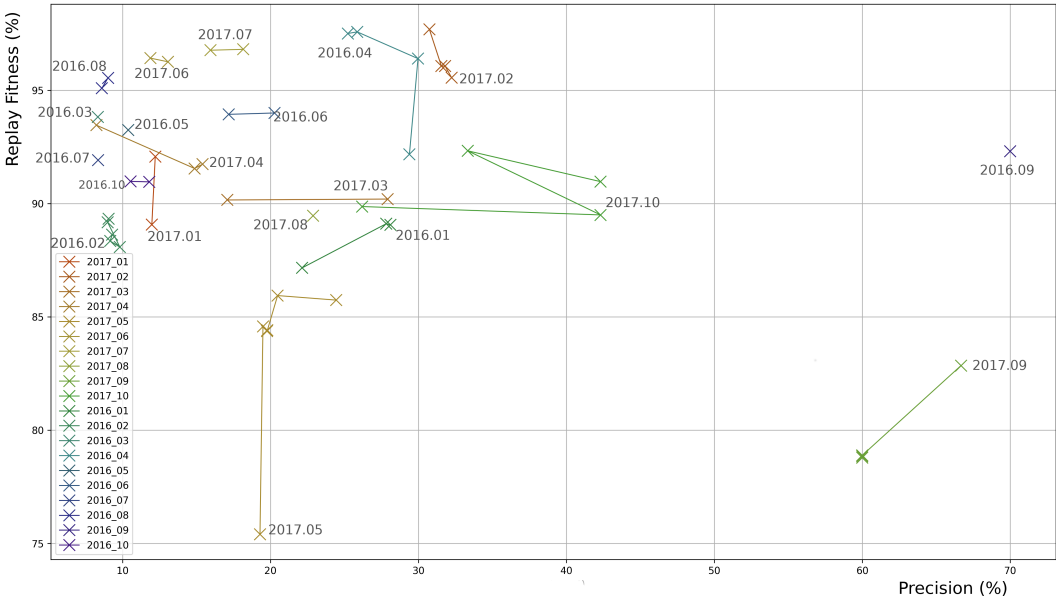


Figure 5.10: A scatter plot showing the relationship between the fitness measure and the precision measure. Once we find a solution, it does not seem to be substituted by an extremely better one. In most of the cases, the Petri nets generated seem to be always close in terms of results to one another.

The original Bergenthum's formulation includes all prefixes of all observed variants into the model, without considering their frequency. Empirically, algorithms derived from this original formulation tend to be overfitting and difficult to read. Our result shown in section 5 presented the same problem.

Usually, in process discovery, we do not try to make sense from complete random sequences. There are real processes, industries, practices generating them. Therefore, we can make an assumption there is an unknown distribution generating these sequences and this distribution may not be stationary through time.

Filtering rare variants is a common strategy to avoid taking into consideration infrequent behavior. This approach is flawed. Maybe there are instances in which various infrequent variants carry essential patterns. Maybe all variants occur just once.

The formulation takes the observed history to describe candidate places which are feasible for all observed history. The process model generated with these places usually is going to serve as an analysis tool or as the model for the process automation for future executions. However, probably all variants observed in history is not observed again in the next period of time.

In the geometrical sense, each variant (distinct sequence) generates a set of its prefixes, which in turn are translated into inequalities. These inequalities are half-spaces in a space described by  $x_i$  and  $y_j$  variables. The union of all these half-spaces describes a polyhedron. Place candidates form bases for the convex set, in the vertices of this polyhedron described by all inequalities generated.

Therefore, suppose that we observe the occurrence of sequences and variants in an online setting. Since we do not expect to observe every possible variant every period of time, the facets of this polyhedron may or may not occur from time to time. Thus, every facet in the polyhedron should carry a measure of uncertainty of their manifestation.

Then all set of constraints associated with a given variant should have a probability of not being taken into considerations. In this new setting, considering that each variant could be manifest itself or not, instead of searching for basic solutions in one polyhedron, we would have to search for

basic solutions in a higher number of polyhedra. If there are  $n$  variants, and each could be "turned on" or "turned off", the number of polyhedra could be as large as  $2^n$  being  $n$  the number of variants.

Finally, we suppose that there is an unknown global distribution which generates the sequences. The observed log is just a sample from this unknown distribution. Consequently, we could establish a probability measure of that specific variant of manifesting itself again proportional to their observed frequency.

## 6.1

### Cardinality constrained Robust Model

The subproblem's feasible set has subsets of inequalities generated by prefixes of the sequences. Each observed case generates a set of inequalities so that the place generated as a solution obeys all of the observed histories. However, what if there are abnormalities in the process, or the data capture itself. Some places can become infeasible whereas other places turn feasible.

In the robust framework, we can model problems with the manifestation of uncertainty in their feasible set (the polyhedron described by the constraints). The robust framework tries to navigate the space of solutions in which there is uncertainty in the feasible boundaries of the problem, being aware of the trade-off between the probability of a solution being feasible and the chance of improving the objective function.

An entirely risk-averse approach would consider all possible constraints conceivable to solve the problem and guarantee that whatever the solution, it is always feasible. This approach frequently produces over-protective solutions. In our problem, a cluttered problem (sequences with many inversions) often produces places with an unpractical number of arcs.

Bertsimas and Sim in (Bertsimas2004) argue for a new kind of modeling uncertainty in the feasible set, imagining a theoretical game played with an adversary. In our case, the simplest and most natural way of modeling is an adversary who has to choose to generate a fixed number of sequences from the unknown distribution. The modeling changes that we need to guarantee that the solution produced should be able to be feasible for at least  $\eta$  percent of the sequences of events observed in history.

Under these assumptions, we deem the original model of the trace constraints to be too risk-averse. In a sense, if there is a possibility of some outlier occurring, probably it does not repeat itself. Additionally, if there are new outliers sequences produced, they probably are different from previous outliers. Therefore, we should provide the model with some flexibility to ignore



a percentage of the observed history.

We have a log  $\mathcal{L}$  composed of  $n$  sequences/variants  $\sigma_i (\forall i \in [n])$ . We define as notation an over-line upon the variable as denoting the prefix-closed set of it:

$$\overline{\sigma_i} = \{\sigma_j | \exists \sigma_k \in A^* | \sigma_j \cdot \sigma_k = \sigma_i\}$$

or the union of the prefix-closed sets of its components:

$$\overline{\mathcal{L}} = \{\sigma_1 \in A^* | \exists \sigma_2 \in A^* | (\sigma_1 \cdot \sigma_2 \in \mathcal{L})\} = \{\overline{\sigma_1} \cup \overline{\sigma_2} \cup \dots \cup \overline{\sigma_n}\}$$

The equivalent formulation must allow for the program the flexibility to shut on and shut off for navigating different execution of scenarios. In order to do that, we introduce additional binary variables  $s_i \in \{0, 1\}$  for each of the variants  $\sigma_i$ . We want to model them as indicators of whether we are exploring trace  $\sigma_i$  or not. Then we define as a convention that if the value of  $s_i$  is 1, we consider the associated inequalities generated by the variant  $\sigma_i$  and, alternatively, we relax these same constraints if  $s_i$  ever has value zero.

In order to do that, we introduce additional binary variables  $s_i \in \{0, 1\}$  for each of the variants  $\sigma_i$ . We want to model them as indicators of whether we are exploring trace  $\sigma_i$  or not. Then we define as a convention that if the value of  $s_i$  is 1, we consider the associated inequalities generated by the variant  $\sigma_i$  and, alternatively, we relax these same constraints if  $s_i$  ever has value zero.

And to each trace constraint we add a term  $M(1 - s_i)$  as to relax the constraint whenever the  $s_i$  associated with the  $\sigma_i$  that generated that constraint. We will denote by  $\vec{p}(\sigma, \mathbf{X}, \mathbf{Y})$  as the function that produces the expression used in the inequality related to the sequence/prefix  $\sigma$

$$\vec{p}(\sigma_j, \mathbf{X}, \mathbf{Y}) + M(1 - s_i) \geq 0 \quad \forall \sigma_j \in \overline{\sigma_i}$$

For this relaxation to be efficient, we set up a tight big  $M$ , equal to the number of variables already in the original inequality. In this manner, whenever the binary variable  $s_i$  has value zero, the inequality is relaxed by the forced introduction of a big  $M$  as a slack.

Each sequence  $\sigma_i$  has a frequency of observation  $f(\sigma_i)$ . This means that for a bag of traces  $B(\mathcal{L}) = \{\sigma_1^{f_1}, \sigma_2^{f_2}, \dots, \sigma_n^{f_n}\}$  with the sequences  $\sigma_i$  and their respective frequencies of observation  $f_i$  we want a model that allows for generated places to be infeasible for some few infrequent variants. Then in the formulation of the sub-problem we add the following constraint:

$$\sum_{i=1}^n f_i s_i \geq \eta \cdot \sum_{i=1}^n f_i \quad (6-1)$$

This constraint that the only valid values for the  $s_i$  values are just for the ones which consider a combination of variants with frequencies above an  $\eta$  proportion of the overall number of cases. This constraint reinforces the search into a space of

To further clarify, please consider the example below.

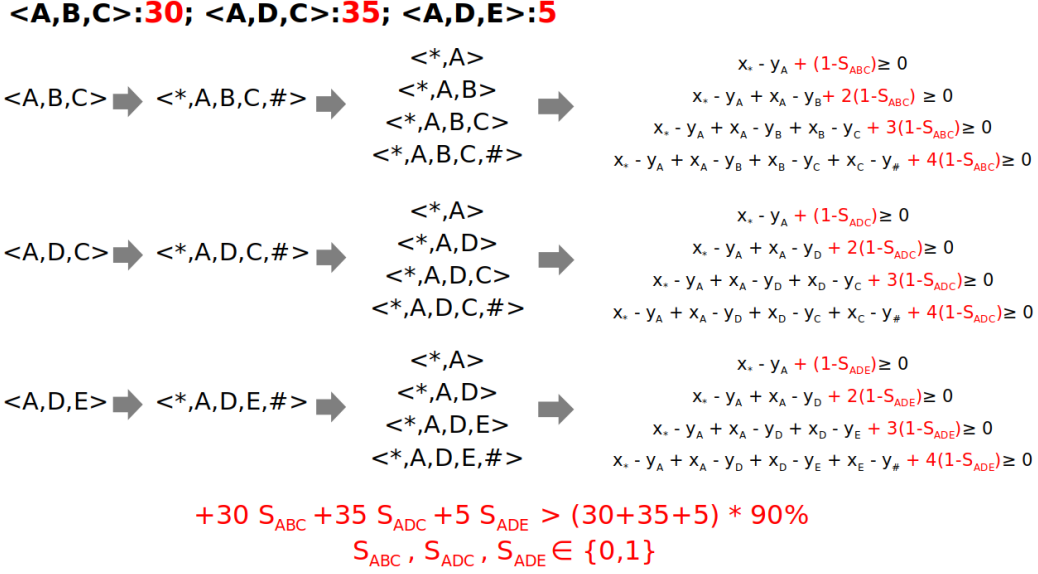


Figure 6.1: Example of the additions made to the previous sub-problem in the branch-cut-and-price schema. For each prefix constraint belonging to the same variant, we add a binary variable that relaxes the problem whenever its value changes from one to zero. An additional constraint is added to control which of the new relaxation binary variables can assume the value 0. The idea is to allow rare variants to be more likely to be shut down. In this particular example, we only allow for the constraints associated with  $\langle A,D,E \rangle$  to be relaxed.

This model factors in a previously not used information: the frequency of observations. Depending on the choice of  $\eta$ , new configurations of the  $s_i$  values are feasible and, thus new possibilities of shutting off infrequent become feasible.

Finally, we must remember a rather obvious result that has practical implications. When we find a solution for a problem with a specific  $\eta_1$ , this solutions is also feasible for any  $\eta_2$  such that  $\eta_2 \leq \eta_1$ . Observe that in equation 6-1, the right-hand side is a constant one we define the data set and desired  $\eta$ . If in the middle of the execution we diminish  $\eta$ , all previous solutions are still feasible. That means, if  $\eta_1 \geq \eta_2$ , then by transitivity:

$$\sum_{i=1}^n f_i s_i \geq \eta_1 \cdot \sum_{i=1}^n f_i \geq \eta_2 \cdot \sum_{i=1}^n f_i \quad (6-2)$$

Therefore, we are allowed to use lower values of  $\eta$  during the execution of the algorithm. Then we can use already found columns with higher  $\eta$ 's to solve smaller  $\eta$  experiments.

## 6.2

### Experiments

The experimentation is very similar to what we did in section 5. We perform the same tests on the same instances and evaluate the leaving-arcs precision and token-replay fitness. Although, in a preliminary attempt of the tests we perceived that the relaxation of  $\eta$  increases the time of executing the sub-problem by a large factor. For example, some instances that took less than 20 minutes using all the constraints, could take days to finish the search for integer solutions.

We observed that solving the problems for  $\eta \in [0.7, 0.9]$  increases execution times for the procedure dramatically. Depending on the instance it peaks in execution time in different values for  $\eta$ . Just for illustration purposes, we present some of the early tests using the event log 7 of the Process Discovery contest of 2017 (PDC2017) in figure 6.2.

eta	time of completion	u	OF
0,9999	0,01 h	8400	
0,95	0,12 h	8370	
0,9	0,23 h	8340	
0,85	1,05 h	8330	
0,8	2,13 h	8320	
0,75	5,75 h	8320	
0,7	14,92 h	8300	
0,65	8,34 h	8290	
0,6	1,89 h	8280	

Figure 6.2: Table showing the execution time and Objective function for the execution of the algorithm using different values for eta. Notice that the execution time peaks at an  $\eta = 0.7$ . Also, our objective function decreases 10 units when a model has one less arc relative to the others. We observe that the opportunity to reduce the quantity of arcs is relatively little by decreasing values from  $\eta$  (eta).

Additionally, we observed that once we find a solution in a particular experiment, we expect the following solutions to achieve no significant improvement. Therefore, since our objective is to observe the effects of decreasing  $\eta$  in the fitness and precision measures, we set up the experiment such that we test using decreasing  $\eta$  values  $\in \{0.9999, 0.9, 0.8, 0.7, 0.6, 0.5\}$  using 12 in-

stances that were able to finish in less than 90 minutes the branching in the last chapter.

The order of the values for  $\eta$  is decreasing such that solving the master model is faster and more stable using the columns already generated for higher values of  $\eta$ . Also, we define a time limit of 120 minutes or two hours for each  $\eta$ , to be able to test all values for  $\eta$  in all the 12 instances.

The 12 instances are: 2017.02, 2017.04, 2017.06, 2017.07, 2017.08, 2017.09, 2017.10, 2016.03, 2016.04, 2016.08, 2016.09, 2016.10.

### 6.3 Results

We present the results in figure 6.3. As expected, the time limit prevented to find solutions in some instances. However, our objective is to see a trend in fitness and precision subject to different values of  $\eta$  decrease, that is, become more risk-prone.

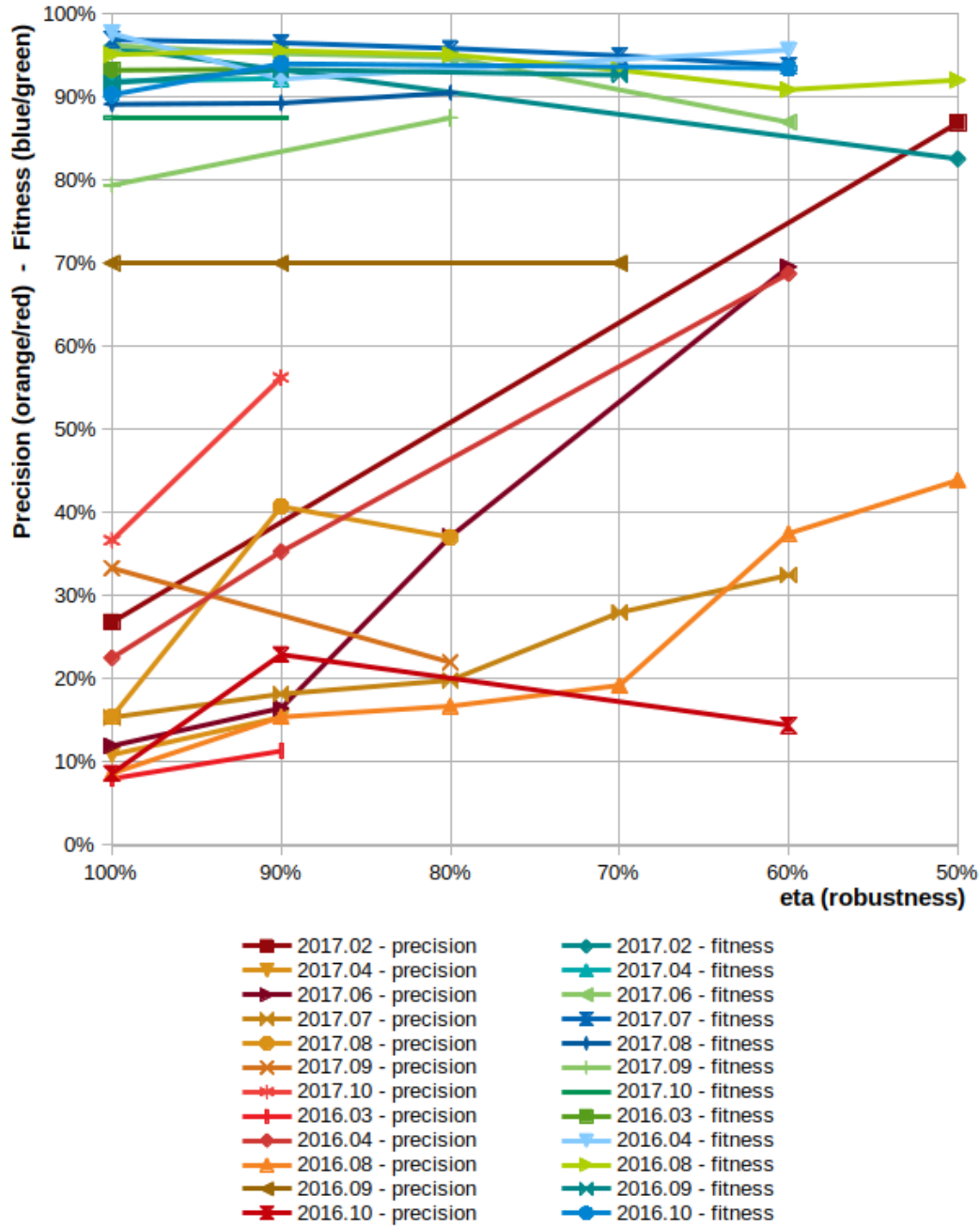


Figure 6.3: Fitness and Precision measures of the 12 instances as  $\eta$  decreases. An  $\eta$  (eta) of 100% on the left is analogous to results obtained in section 5. We can observe the progressive relaxation of  $\eta$  as the lines of the chart move to the right. A counter-intuitive result: generating places and arcs allowing the model to ignore a large amount of history can yield better precision while maintaining high fitness.

Also, the models tend to be easier to read. Since the objective function guides the search for solutions with fewer arcs, we expect that relaxing the problem would yield simpler Petri nets. In Appendix B, we can observe the processes rendered.

By looking at them, we observe that some instances have a significant

gain in visual simplicity as the parameter  $\eta$  declines. Even in the instances in which this gain is subtle, the addition of new places substitute some cycles and parallelism by a more streamlined version of a specific part of the process.

## 7

## Discussion

### 7.1

#### Findings and contributions

As expected the model does not handle situations which silent ( $\tau$ ) activities are required. However, we noticed that the process model could still play-out the log, although the process model is not sound. This means that tokens are left in intermediary places after the Petri net finishes its execution.

In the compact global model, the algorithm did not seem to handle well parallelism. The main problem with all the wrong tests was that the model bifurcates streams correctly using an AND-gate, but it closes them using an OR-gate, which may cause problems with the Petri net's soundness. The constraint which obligated the sum of all leaving arcs to be equal to the sum of entering arcs for all intermediary places seemed to work well in these cases.

Cases in which there were only OR-gates, all variants seemed to perform well. All the cases, 3, 4, 6, 7 and 10 returned the right answer consistently. One particular characteristic of the OR-gate is that it does not change the total amount of tokens when activated.

Variants that try different schemes for objective function had a great sensibility in the correctness of the result. Among the base case and variants 1, 2 and 3, the variants using mixed weights in both W and XY or Z and XY families got the best results. Just as in the ILP case, there is still room for further research on the balance of objective function weights.

In the Branch-cut-and-price algorithm, the gains in performance were vast. It was possible to handle larger instances that were created intentionally cluttered. Also, we do not need to guess a  $K$  to make the problem faster or slower; it is adaptable. Furthermore, the lazy introduction of place candidates reduces in a great deal the number of combinatorial interactions that integer variables from different blocks should handle.

We observed that there is not a steady gain of waiting for the branching to terminate. Solutions usually remain with the same characteristics of the first solution found. Also, the current master objective does not seem to

promote substantial improvements in the fitness measure. On the contrary for the precision measure, the algorithm while finding solutions with fewer arcs seems to improve significantly precision. When we further allowed the search for places to ignore part of the history, theoretically we relaxed the problem in a way that it could find solutions with an even lower objective function, thus fewer arcs. This strategy yielded Petri nets with a substantial gain in the precision metric without losing too much on the fitness metric.

Finally, we observed that the relaxation in the production of places could yield simpler preciser processes. However, there is a significant increase in the search space for a solution, so that the MIP solver used a considerable additional time to solve the subproblem. The remaining process models can be found in the appendices of this dissertation or here in the Dropbox folder. We invite the reader to check the perceived gain in simplicity in other instances.

For example, we compare results for instances in Log6 and Log7 from the Process Discovery Contest of 2017 for  $\eta = 0.9999$  and  $\eta = 0.6$ . The gains in perceived simplicity were very noticeable. However, the gain in the precision metric was high only with Log6. This shows that the gain in perceived simplicity is possible even when the gain in other metrics was not present. Also, we can observe that structures inducing exclusive decisions substitute many substructures that generate parallelism. This may be the motive for the observed gain in precision.

## 7.2

### Limitations and future research

We believe this algorithm to be a work in progress. We showed in this work that there is potential to achieve versions of this algorithm that could compete against other state-of-the-art algorithms. The current limitations of the algorithm, still to be worked upon, are:

- disconnection of the objective function and the improvement of measures such as fitness and precision;
- model makes assumptions such as not generating duplicates of transitions that generate the same symbol when firing and not using silent  $\tau$  activities, these assumptions reduce the capacity of the chosen representation (Petri nets) to be fully expressive;
- there is also a considerable integrality gap and need for branching that slows the algorithm;
- the relaxation of the place generation in the risk-prone modeling can have a big toll on the processing time of the subproblem.



Therefore, to mitigate the enlisted problems, we believe that future research could work on: a better Global model, better polyhedral understanding of the global model, new search strategies for integer solutions, using the information obtained evaluating and simulating intermediary solutions obtained. We detail these next.

We need to better understand how the constraints and the proportions of weights in the objective function may affect other aspects of the process models we did not approach in this work such as soundness of the model. Moreover, we observed that fixing the start and finish reducing the optimization time. We believe that there are opportunities in discovering heuristics that allow the model at the to fix more variables, especially the set of  $w_{ij}$  variables.

The model could also be further improved to use silent activities  $\tau$ . Silent transitions would be useful for modeling skips, some nested cycles, and compositions of gates such as AND-gates followed by OR-gates. Any advance on this sense could improve the representational power of this new approach, and further gains in precision.

We have an original formulation which generates candidates of places for Petri nets that a Master program connects. Since the most basic version of the formulation is totally unimodular, we believe that there is an opportunity for developing a primal-dual algorithm for generating batches of feasible places. The fast generation could yield a strategy of adding a large number of columns from the start to avoid the depth of branching.

Additionally, the token-replay fitness metric and the leaving arcs precision can be broken-down as regional indicators of nonconformity to direct us to subsets of variables which still have to improve. Maybe that there is an opportunity to use this information for intensification in a meta-heuristic search for new place alternatives.

There are lots of opportunities to study the branch-cut-and-price algorithm using classic column-generation algorithm strategies. Adaptive column generation stabilization, new branching strategies (i.e., leaf branches enumeration), strong branching, reduced cost fixation, to name a few.

The integrality gap between 1% and 5% shows that there may be an opportunity for studying the master polytope and proposing specialized valid cuts to approximate the linear relaxation of the master to the integer solutions. Then we can improve the Cutting oracle aspect of the algorithm in order to reduce the depth needed for the branching.

Additionally, we showed that the relaxation of the  $\eta$  parameter tends to significantly improve the time to solve. Our current approach of finding an optimal reduced cost in the sub-problem may be flawed. We should solve for

all feasible combinations and add all the places of all feasible combinations of turning on and off constraints using the added integer variables  $s_i$ . Just finding the most negative solution of all possible combinations, and there are a lot of possible combinations, is inefficient.

## Bibliography

- [Adriansyah2015] ADRIANSYAH, A.; MUNOZ-GAMA, J.; CARMONA, J.; VAN DONGEN, B. F. ; VAN DER AALST, W. M.. **Measuring precision of modeled behavior**. Information systems and e-Business Management, 13(1):37–67, 2015.
- [Bergenthum2007] BERGENTHUM, R.; DESEL, J.; LORENZ, R. ; MAUSER, S.. **Process mining based on regions of languages**. In: INTERNATIONAL CONFERENCE ON BUSINESS PROCESS MANAGEMENT, p. 375–383. Springer, 2007.
- [Bertsimas2004] BERTSIMAS, D.; SIM, M.. **The price of robustness**. Operations research, 52(1):35–53, 2004.
- [Carmona2010] CARMONA, J.; CORTADELLA, J.. **Process mining meets abstract interpretation**. In: JOINT EUROPEAN CONFERENCE ON MACHINE LEARNING AND KNOWLEDGE DISCOVERY IN DATABASES, p. 184–199. Springer, 2010.
- [Carmona2018] CARMONA, J.; VAN DONGEN, B.; SOLTÍ, A. ; WEIDLICH, M.. **Conformance Checking: Relating Processes and Models**. Springer, 2018.
- [Chvátal1983] CHVATAL, V.; CHVATAL, V. ; OTHERS. **Linear programming**. Macmillan, 1983.
- [Conejo2006] CONEJO, A. J.; CASTILLO, E.; MINGUEZ, R. ; GARCIA-BERTRAND, R.. **Decomposition techniques in mathematical programming: engineering and science applications**. Springer Science & Business Media, 2006.
- [Conforti2014] CONFORTI, M.; CORNUÉJOLS, G. ; ZAMBELLI, G.. **Integer programming models**. In: INTEGER PROGRAMMING, p. 45–84. Springer, 2014.
- [Dantzig1960] DANTZIG, G. B.; WOLFE, P.. **Decomposition principle for linear programs**. Operations research, 8(1):101–111, 1960.

- [Ekanayake2013] EKANAYAKE, C. C.; DUMAS, M.; GARCÍA-BAÑUELOS, L. ; LA ROSA, M.. **Slice, mine and dice: Complexity-aware automated discovery of business process models**. In: BUSINESS PROCESS MANAGEMENT, p. 49–64. Springer, 2013.
- [Ford1958] FORD JR, L. R.; FULKERSON, D. R.. **A suggested computation for maximal multi-commodity network flows**. Management Science, 5(1):97–101, 1958.
- [Fortet1960] FORTET, R.. **Applications de l’algebre de boole en recherche opérationelle**. Revue Française de Recherche Opérationelle, 4(14):17–26, 1960.
- [Leemans2013] LEEMANS, S. J.; FAHLAND, D. ; VAN DER AALST, W. M.. **Discovering block-structured process models from event logs containing infrequent behaviour**. In: INTERNATIONAL CONFERENCE ON BUSINESS PROCESS MANAGEMENT, p. 66–78. Springer, 2013.
- [Leemans2014] LEEMANS, S. J.; FAHLAND, D. ; VAN DER AALST, W. M.. **Discovering block-structured process models from incomplete event logs**. In: INTERNATIONAL CONFERENCE ON APPLICATIONS AND THEORY OF PETRI NETS AND CONCURRENCY, p. 91–110. Springer, 2014.
- [Lorenz2007HowTS] LORENZ, R. C.; MAUSER, S. ; JUHÁS, G.. **How to synthesize nets from languages - a survey**. 2007 Winter Simulation Conference, p. 637–647, 2007.
- [Munoz2010] MUÑOZ-GAMA, J.; CARMONA, J.. **A fresh look at precision in process conformance**. In: INTERNATIONAL CONFERENCE ON BUSINESS PROCESS MANAGEMENT, p. 211–226. Springer, 2010.
- [PDC2016] **Process Discovery Contest 2016**. [https://www.win.tue.nl/ieeetfpm/doku.phpid=shared:edition\\_2016](https://www.win.tue.nl/ieeetfpm/doku.phpid=shared:edition_2016). Last accessed: 2019-02-20.
- [PDC2017] **Process Discovery Contest 2017**. [https://www.win.tue.nl/ieeetfpm/doku.phpid=shared:process\\_discovery\\_contest](https://www.win.tue.nl/ieeetfpm/doku.phpid=shared:process_discovery_contest). Last accessed: 2019-02-20.
- [Reisig2013] REISIG, W.. **Understanding petri nets: modeling techniques, analysis methods, case studies**. Springer, 2013.

- [Rozinat2007] ROZINAT, A.; DE MEDEIROS, A. A.; GÜNTHER, C. W.; WEIJTERS, A. ; VAN DER AALST, W. M.. **Towards an evaluation framework for process mining algorithms**. BPM Center Report BPM-07-06, BPMcenter. org, 123:142, 2007.
- [VanDerAalst2011] VAN DER AALST, W. M.. **Process discovery: An introduction**. In: PROCESS MINING, p. 125–156. Springer, 2011.
- [VanDerAalst2013] VAN DER AALST, W. M.. **Decomposing petri nets for process mining: A generic approach**. Distributed and Parallel Databases, 31(4):471–507, 2013.
- [VanDerAalst2016] VAN DER AALST, W.. **Process Mining: Data Science in Action**. Springer Berlin Heidelberg, 2016.
- [VanDerWerf2008] VAN DER WERF, J. M. E.; VAN DONGEN, B. F.; HURKENS, C. A. ; SEREBRENIK, A.. **Process discovery using integer linear programming**. In: INTERNATIONAL CONFERENCE ON APPLICATIONS AND THEORY OF PETRI NETS, p. 368–387. Springer, 2008.
- [VanDerWerf2009] VAN DER WERF, J. M. E. M.; VAN DONGEN, B. F.; HURKENS, C. A. J. ; SEREBRENIK, A.. **Process discovery using integer linear programming**. Fundam. Inf., 94(3-4):387–412, Aug. 2009.
- [VanZelst2015] VAN ZELST, S. J.; VAN DONGEN, B. F. ; VAN DER AALST, W. M.. **Avoiding over-fitting in ilp-based process discovery**. In: INTERNATIONAL CONFERENCE ON BUSINESS PROCESS MANAGEMENT, p. 163–171. Springer, 2015.
- [VanZelst2015] VAN ZELST, S. J.; VAN DONGEN, B. F. ; VAN DER AALST, W. M.. **Ilp-based process discovery using hybrid regions**. In: ATAED@ PETRI NETS/ACSD, p. 47–61, 2015.
- [Vanderbeck2005] VANDERBECK, F.. **Implementing mixed integer column generation**. In: COLUMN GENERATION, p. 331–358. Springer, 2005.
- [Verbeek2014] VERBEEK, H.; VAN DER AALST, W. M.. **Decomposed process mining: The ilp case**. In: INTERNATIONAL CONFERENCE ON BUSINESS PROCESS MANAGEMENT, p. 264–276. Springer, 2014.
- [Verbeek2017] VERBEEK, H.; VAN DER AALST, W. ; MUNOZ-GAMA, J.. **Divide and conquer: a tool framework for supporting decomposed**

discovery in process mining. The Computer Journal, 60(11):1649–1674, 2017.

[Wentges1997] WENTGES, P.. **Weighted dantzig–wolfe decomposition for linear mixed-integer programming.** International Transactions in Operational Research, 4(2):151–162, 1997.



## A

### BCP model computational experiment detailed

case	event	minutes	precision	fitness	Gap
2017_01	begin	0,00			
2017_01	root	4,19			
2017_01	IntSol1	9,09	11,98%	89,07%	0,84%
2017_01	IntSol2	15,35	12,21%	92,07%	0,79%
2017_02	begin	0,00			
2017_02	root	1,77			
2017_02	IntSol1	2,52	30,74%	97,69%	0,32%
2017_02	IntSol2	3,80	31,55%	96,07%	0,27%
2017_02	IntSol3	3,98	31,80%	96,07%	0,22%
2017_02	IntSol4	5,22	32,24%	95,56%	0,12%
2017_02	finish	6,43			
2017_03	begin	0,00			
2017_03	root	1,21			
2017_03	IntSol1	2,41	17,08%	90,16%	1,09%
2017_03	IntSol2	4,94	27,92%	90,20%	1,02%
2017_03	finish	107,30			
2017_04	begin	0,00			
2017_04	root	1,99			
2017_04	IntSol1	3,89	8,24%	93,46%	1,16%
2017_04	IntSol2	7,73	14,87%	91,55%	1,09%
2017_04	IntSol3	21,67	15,41%	91,74%	1,01%
2017_04	finish	30,83			
2017_05	root	5,11			
2017_05	IntSol1	15,35	24,44%	85,74%	1,22%
2017_05	IntSol2	16,76	20,48%	85,94%	1,16%
2017_05	IntSol3	19,72	19,77%	84,37%	1,11%
2017_05	IntSol4	26,02	19,78%	84,41%	1,06%
2017_05	IntSol5	85,71	19,49%	84,58%	1,00%
2017_05	IntSol6	144,42	19,29%	75,40%	0,95%
2017_06	begin	0,00			
2017_06	root	0,47			
2017_06	IntSol1	0,82	13,07%	96,26%	0,89%
2017_06	IntSol2	1,27	11,89%	96,42%	0,82%
2017_06	finish	2,70			
2017_07	begin	0,00			
2017_07	root	0,09			
2017_07	IntSol1	0,17	15,94%	96,77%	1,57%
2017_07	IntSol2	0,51	18,16%	96,81%	1,45%
2017_07	finish	1,60			
2017_08	begin	0,00			
2017_08	root	0,29			
2017_08	IntSol1	0,45	22,87%	89,46%	0,58%
2017_08	finish	85,35			
2017_09	begin	0,00			
2017_09	root	1,20			
2017_09	IntSol1	1,81	60,00%	78,78%	0,85%
2017_09	IntSol2	2,05	60,00%	78,85%	0,79%
2017_09	IntSol3	9,68	60,00%	78,89%	0,74%
2017_09	IntSol4	39,74	66,67%	82,85%	0,71%
2017_09	finish	56,65			
2017_10	begin	0,00			
2017_10	root	0,38			
2017_10	IntSol1	0,85	26,19%	89,87%	0,99%
2017_10	IntSol2	3,61	42,31%	89,50%	0,91%
2017_10	IntSol3	15,66	33,33%	92,33%	0,87%
2017_10	IntSol4	18,31	42,31%	90,97%	0,83%
2017_10	finish	82,06			



case	event	minutes	precision	fitness	Gap
2016_01	begin	0,00			
2016_01	root	1,03			
2016_01	IntSol1	2,19	22,13%	87,16%	1,26%
2016_01	IntSol2	591,50	27,82%	89,11%	1,18%
2016_01	IntSol3	597,80	28,10%	89,06%	1,02%
2016_01	finish	649,09			
2016_02	begin	0,00			
2016_02	root	3,32			
2016_02	IntSol1	11,36	9,05%	89,33%	2,45%
2016_02	IntSol2	15,67	8,99%	89,18%	2,37%
2016_02	IntSol3	18,05	9,82%	88,08%	2,29%
2016_02	IntSol4	19,22	9,16%	88,35%	2,21%
2016_02	IntSol5	24,35	9,30%	88,63%	2,14%
2016_03	begin	0,00			
2016_03	root	0,97			
2016_03	IntSol1	1,47	8,33%	93,82%	0,20%
2016_03	finish	35,43			
2016_04	begin	0,00			
2016_04	root	0,37			
2016_04	IntSol1	0,74	25,23%	97,51%	0,88%
2016_04	IntSol2	0,93	25,86%	97,58%	0,81%
2016_04	IntSol3	1,29	29,97%	96,40%	0,74%
2016_04	IntSol4	3,50	29,39%	92,18%	0,61%
2016_04	finish	24,68			
2016_05	begin	0,00			
2016_05	root	66,45			
2016_05	IntSol1	282,31	10,39%	93,25%	2,37%
2016_06	begin	0,00			
2016_06	root	2,28			
2016_06	IntSol1	5,26	17,17%	93,94%	2,12%
2016_06	IntSol2	32,75	20,27%	94,00%	2,04%
2016_06	finish	174,23			
2016_07	begin	0,00			
2016_07	root	69,74			
2016_07	IntSol1	229,64	8,35%	91,92%	1,76%
2016_08	begin	0,00			
2016_08	root	0,46			
2016_08	IntSol1	0,79	8,60%	95,09%	0,90%
2016_08	IntSol2	2,22	9,02%	95,55%	0,82%
2016_08	finish	16,70			
2016_09	begin	0,00			
2016_09	root	0,41			
2016_09	IntSol1	0,79	70,00%	92,31%	1,13%
2016_09	finish	10,68			
2016_10	begin	0,00			
2016_10	root	0,71			
2016_10	IntSol1	1,24	10,54%	90,98%	1,03%
2016_10	IntSol2	2,15	11,80%	90,96%	0,96%
2016_10	finish	16,55			



## B

## Risk-prone model computational experiment detailed

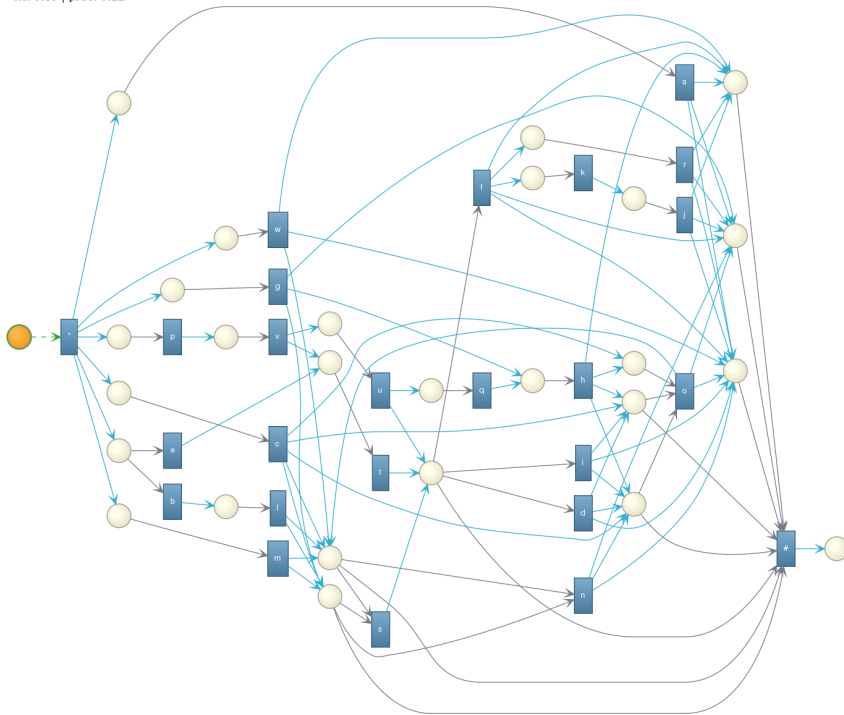
case	eta	event	precision	fitness	Integer OF	Root OF	Gap	hours
2017.02		begin						0.0
2017.02	100%	root			20036,671	20036,671		0.0
2017.02	100%	IntSol1	26,83%	97,44%	20090,015	20036,671	0,27%	0.0
2017.02	100%	IntSol2	30,00%	97,45%	20080,010	20036,671	0,22%	0.0
2017.02	100%	IntSol3	31,32%	95,93%	20070,010	20036,671	0,17%	0.0
2017.02	100%	IntSol4	30,87%	95,94%	20060,005	20036,671	0,12%	0.1
2017.02	100%	finish			20060,005	20036,671	0,12%	0.1
2017.02	90%	root			19998,343	19998,343		0.1
2017.02	90%	finish			20060,005	19998,343	0,31%	0.5
2017.02	80%	root			19998,343	19998,343		0.5
2017.02	80%	finish			20060,005	19998,343	0,31%	1.7
2017.02	70%	root			19998,343	19998,343		1.7
2017.02	70%	finish			20060,005	19998,343	0,31%	3.1
2017.02	60%	root			19988,338	19988,338		3.1
2017.02	50%	root			19988,338	19988,338		5.2
2017.02	50%	IntSol1	86,89%	82,55%	20050,015	19988,338	0,31%	5.2
2017.04		begin						0.0
2017.04	100%	root			12900,024	12900,024		0.0
2017.04	100%	IntSol1	10,88%	93,70%	13040,083	12900,024	1,09%	0.1
2017.04	100%	IntSol2	17,01%	91,88%	13030,073	12900,024	1,01%	0.1
2017.04	100%	finish			13030,073	12900,024	1,01%	0.3
2017.04	90%	root			12884,181	12884,181		0.6
2017.04	90%	IntSol1	15,40%	92,22%	13000,047	12884,181	0,90%	1.4
2017.04	80%	root			12868,341	12868,341		3.5
2017.04	70%	root			12863,338	12863,338		13.6
2017.04	60%	root			12850,005	12850,005		15.4
2017.04	50%	root			12845,005	12845,005		32.2
2017.06		begin						0.0
2017.06	100%	root			14401,673	14401,673		0.0
2017.06	100%	IntSol1	15,48%	96,16%	14540,053	14401,673	0,96%	0.0
2017.06	100%	IntSol2	12,11%	96,37%	14530,043	14401,673	0,89%	0.0
2017.06	100%	IntSol3	11,95%	96,42%	14520,039	14401,673	0,82%	0.0
2017.06	100%	finish			14520,039	14401,673	0,82%	0.0
2017.06	90%	root			14342,338	14342,338		0.1
2017.06	90%	IntSol1	16,49%	95,20%	14460,045	14342,338	0,82%	0.1
2017.06	90%	IntSol2	25,20%	95,77%	14450,030	14342,338	0,75%	1.6
2017.06	80%	root			14332,338	14332,338		2.1
2017.06	80%	IntSol1	37,12%	94,75%	14440,005	14332,338	0,75%	4.0
2017.06	70%	root			14319,000	14319,000		4.1
2017.06	60%	root			14319,000	14319,000		6.1
2017.06	60%	IntSol1	69,57%	86,95%	14430,000	14319,000	0,78%	7.4
2017.06	50%	root			14319,000	14319,000		8.1
2017.07		begin						0.0
2017.07	100%	root			8280,016	8280,016		0.0
2017.07	100%	IntSol1	15,37%	96,86%	8400,062	8280,016	1,45%	0.0
2017.07	100%	finish			8400,062	8280,016	1,45%	0.0
2017.07	90%	root			8246,673	8246,673		0.0
2017.07	90%	IntSol1	18,19%	96,50%	8340,023	8246,673	1,13%	0.0
2017.07	90%	finish			8340,023	8246,673	1,13%	1.6
2017.07	80%	root			8216,673	8216,673		1.6
2017.07	80%	IntSol1	19,84%	96,29%	8330,035	8216,673	1,38%	1.7
2017.07	80%	IntSol2	21,31%	95,84%	8320,009	8216,673	1,26%	1.7
2017.07	70%	root			8210,003	8210,003		3.7
2017.07	70%	IntSol1	32,49%	94,97%	8310,025	8210,003	1,22%	3.7
2017.07	70%	IntSol2	27,98%	95,36%	8300,019	8210,003	1,10%	3.8
2017.07	60%	root			8200,003	8200,003		5.7
2017.07	60%	IntSol1	32,49%	93,72%	8280,015	8200,003	0,98%	5.7
2017.08		begin						0.0
2017.08	100%	root			12805,833	12805,833		0.0
2017.08	100%	IntSol1	15,44%	90,85%	12940,059	12805,833	1,05%	0.0
2017.08	100%	IntSol2	23,38%	89,11%	12910,039	12805,833	0,81%	0.0
2017.08	100%	IntSol3	19,18%	89,20%	12900,035	12805,833	0,74%	0.0
2017.08	100%	IntSol4	20,41%	89,19%	12890,030	12805,833	0,66%	0.0
2017.08	100%	IntSol5	22,89%	89,96%	12880,010	12805,833	0,58%	0.0
2017.08	100%	finish			12880,010	12805,833	0,58%	1.6
2017.08	90%	root			12800,040	12800,040		1.6
2017.08	90%	IntSol1	40,74%	89,25%	12876,672	12800,040	0,60%	2.3
2017.08	80%	root			12788,056	12788,056		3.8
2017.08	80%	IntSol1	37,03%	90,48%	12870,015	12788,056	0,64%	4.4
2017.08	70%	root			12788,056	12788,056		5.5
2017.08	60%	root			12767,083	12767,083		8.1
2017.08	50%	root			12760,417	12760,417		10.0
2017.09		begin						0.0
2017.09	100%	root			17947,652	17947,652		0.0
2017.09	100%	IntSol1	33,33%	79,38%	18080,065	17947,652	0,74%	0.0
2017.09	100%	IntSol2	33,33%	79,49%	18070,060	17947,652	0,68%	0.1
2017.09	90%	root			17935,017	17935,017		2.2
2017.09	80%	root			17919,300	17919,300		4.1
2017.09	80%	IntSol1	22,00%	87,47%	18060,073	17919,300	0,79%	4.5
2017.09	70%	root			17918,798	17918,798		6.3
2017.09	60%	root			17918,054	17918,054		8.7
2017.09	50%	root			17914,125	17914,125		10.6
2017.10		begin						0.0
2017.10	100%	root			12248,408	12248,408		0.0
2017.10	100%	IntSol1	37,93%	87,53%	12380,085	12248,408	1,08%	0.0
2017.10	100%	IntSol2	36,67%	89,14%	12370,085	12248,408	0,99%	0.1
2017.10	100%	IntSol3	42,31%	89,50%	12360,080	12248,408	0,91%	0.7
2017.10	100%	IntSol4	42,31%	90,97%	12350,065	12248,408	0,83%	1.7
2017.10	90%	root			12243,396	12243,396		2.0
2017.10	90%	IntSol1	56,25%	87,44%	12340,065	12243,396	0,79%	2.2
2017.10	80%	root			12242,564	12242,564		4.1
2017.10	70%	root			12241,729	12241,729		6.1
2017.10	60%	root			12241,729	12241,729		11.0
2017.10	50%	root			12241,652	12241,652		16.4

case	eta	event	precision	fitness	Integer OF	Root OF	Gap	hours
2016.03		begin						0,0
2016.03	100%	root			18033,996	18033,996		0,0
2016.03	100%	IntSol1	8,03%	93,27%	18080,085	18033,996	0,26%	0,0
2016.03	100%	IntSol2	8,00%	93,20%	18070,075	18033,996	0,20%	0,2
2016.03	100%	finish			18070,075	18033,996	0,20%	0,7
2016.03	90%	root			17969,195	17969,195		1,2
2016.03	90%	IntSol1	11,48%	93,62%	18034,080	17969,195	0,36%	1,4
2016.03	90%	IntSol2	11,33%	93,34%	18032,586	17969,195	0,35%	2,0
2016.03	80%	root			17966,330	17966,330		3,5
2016.03	70%	root			17961,918	17961,918		9,1
2016.03	60%	root			17959,745	17959,745		12,1
2016.03	50%	root			17958,117	17958,117		19,7
2016.04		begin						0,0
2016.04	100%	root			14402,805	14402,805		0,0
2016.04	100%	IntSol1	23,45%	99,10%	14530,039	14402,805	0,88%	0,0
2016.04	100%	IntSol2	22,52%	99,13%	14520,034	14402,805	0,81%	0,0
2016.04	100%	IntSol3	27,24%	97,64%	14490,044	14402,805	0,61%	0,1
2016.04	100%	finish			14490,044	14402,805	0,61%	0,2
2016.04	90%	root			14361,667	14361,667		0,2
2016.04	90%	IntSol1	42,75%	97,68%	14480,030	14361,667	0,82%	0,3
2016.04	90%	IntSol2	35,31%	92,15%	14470,034	14361,667	0,75%	0,3
2016.04	80%	root			14361,667	14361,667		2,2
2016.04	70%	root			14361,667	14361,667		4,2
2016.04	60%	root			14356,000	14356,000		6,2
2016.04	60%	IntSol1	68,75%	95,66%	14465,000	14356,000	0,76%	6,2
2016.04	50%	root			14355,000	14355,000		8,2
2016.08		begin						0,0
2016.08	100%	root			13370,185	13370,185		0,0
2016.08	100%	IntSol1	8,64%	95,10%	13490,084	13370,185	0,90%	0,0
2016.08	100%	IntSol2	8,86%	95,58%	13480,083	13370,185	0,82%	0,0
2016.08	100%	IntSol3	9,06%	95,73%	13470,052	13370,185	0,75%	0,2
2016.08	100%	finish			13470,052	13370,185	0,75%	0,4
2016.08	90%	root			13337,713	13337,713		0,4
2016.08	90%	IntSol1	15,42%	95,55%	13430,045	13337,713	0,69%	0,4
2016.08	80%	root			13325,005	13325,005		2,4
2016.08	80%	IntSol1	16,73%	95,07%	13420,035	13325,005	0,71%	2,4
2016.08	70%	root			13313,333	13313,333		4,5
2016.08	70%	IntSol1	19,22%	94,38%	13410,020	13313,333	0,73%	4,8
2016.08	70%	IntSol2	22,09%	94,92%	13405,015	13313,333	0,69%	5,6
2016.08	70%	IntSol3	24,99%	93,23%	13400,005	13313,333	0,65%	5,9
2016.08	60%	root			13305,833	13305,833		6,4
2016.08	60%	IntSol1	37,50%	90,87%	13392,505	13305,833	0,65%	6,7
2016.08	50%	root			13301,667	13301,667		8,6
2016.08	50%	IntSol1	43,90%	92,02%	13390,020	13301,667	0,66%	8,7
2016.09		begin						0,0
2016.09	100%	root			11806,342	11806,342		0,0
2016.09	100%	IntSol1	70,00%	94,33%	11950,034	11806,342	1,22%	0,0
2016.09	100%	IntSol2	70,00%	91,67%	11940,020	11806,342	1,13%	0,1
2016.09	100%	finish			11940,020	11806,342	1,13%	0,2
2016.09	90%	root			11800,983	11800,983		0,2
2016.09	90%	IntSol1	70,00%	93,29%	11930,040	11800,983	1,09%	0,3
2016.09	80%	root			11800,983	11800,983		2,2
2016.09	70%	root			11800,947	11800,947		4,2
2016.09	70%	IntSol1	70,00%	92,64%	11920,025	11800,947	1,01%	4,5
2016.09	60%	root			11800,200	11800,200		6,3
2016.09	50%	root			11799,000	11799,000		8,5
2016.10		begin						0,0
2016.10	100%	root			14382,098	14382,098		0,0
2016.10	100%	IntSol1	8,65%	90,26%	14520,054	14382,098	0,96%	0,0
2016.10	100%	finish			14520,054	14382,098	0,96%	0,3
2016.10	90%	root			14345,575	14345,575		0,5
2016.10	90%	IntSol1	22,93%	93,99%	14460,069	14345,575	0,80%	0,7
2016.10	80%	root			14344,820	14344,820		2,4
2016.10	70%	root			14344,820	14344,820		4,4
2016.10	60%	root			14334,825	14334,825		8,0
2016.10	60%	IntSol1	14,43%	93,44%	14450,050	14334,825	0,80%	9,1
2016.10	50%	root			14334,825	14334,825		10,1

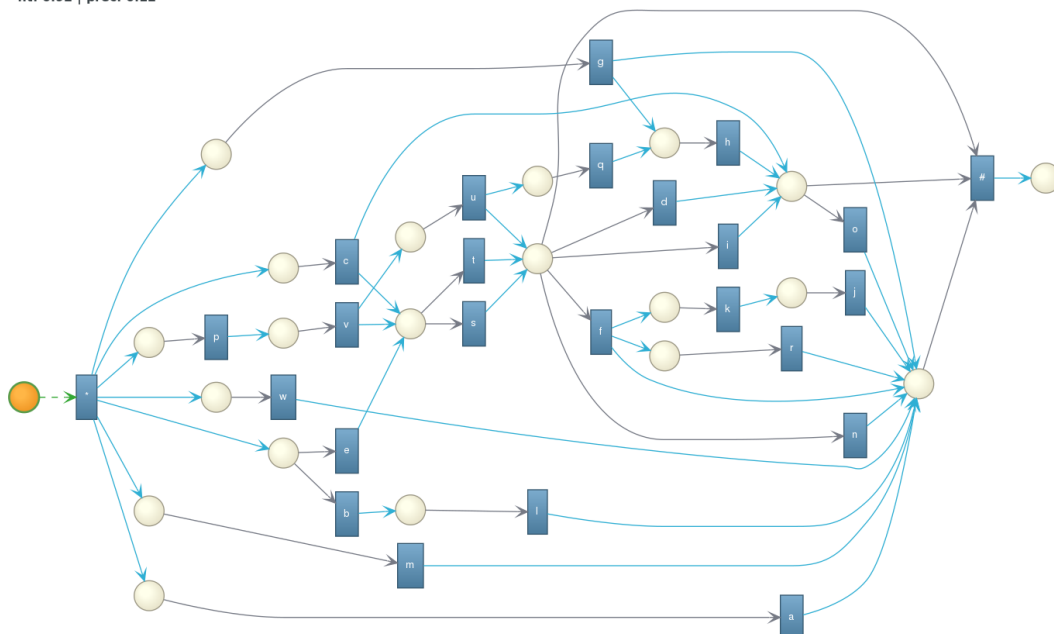
## C

### Process models generated using the Branch-Cut-and-Price Model

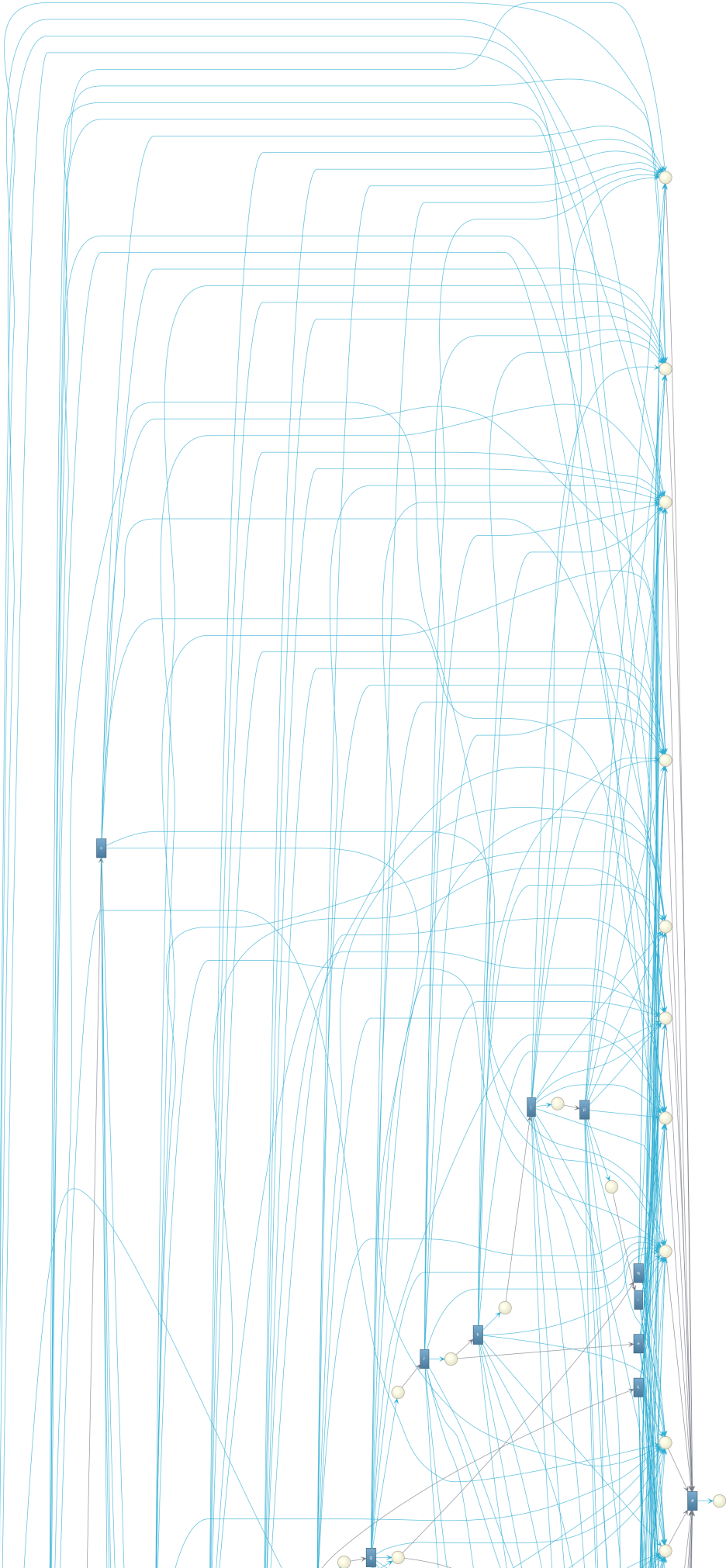
**2017.01**  
fit: 0.89 | prec: 0.12



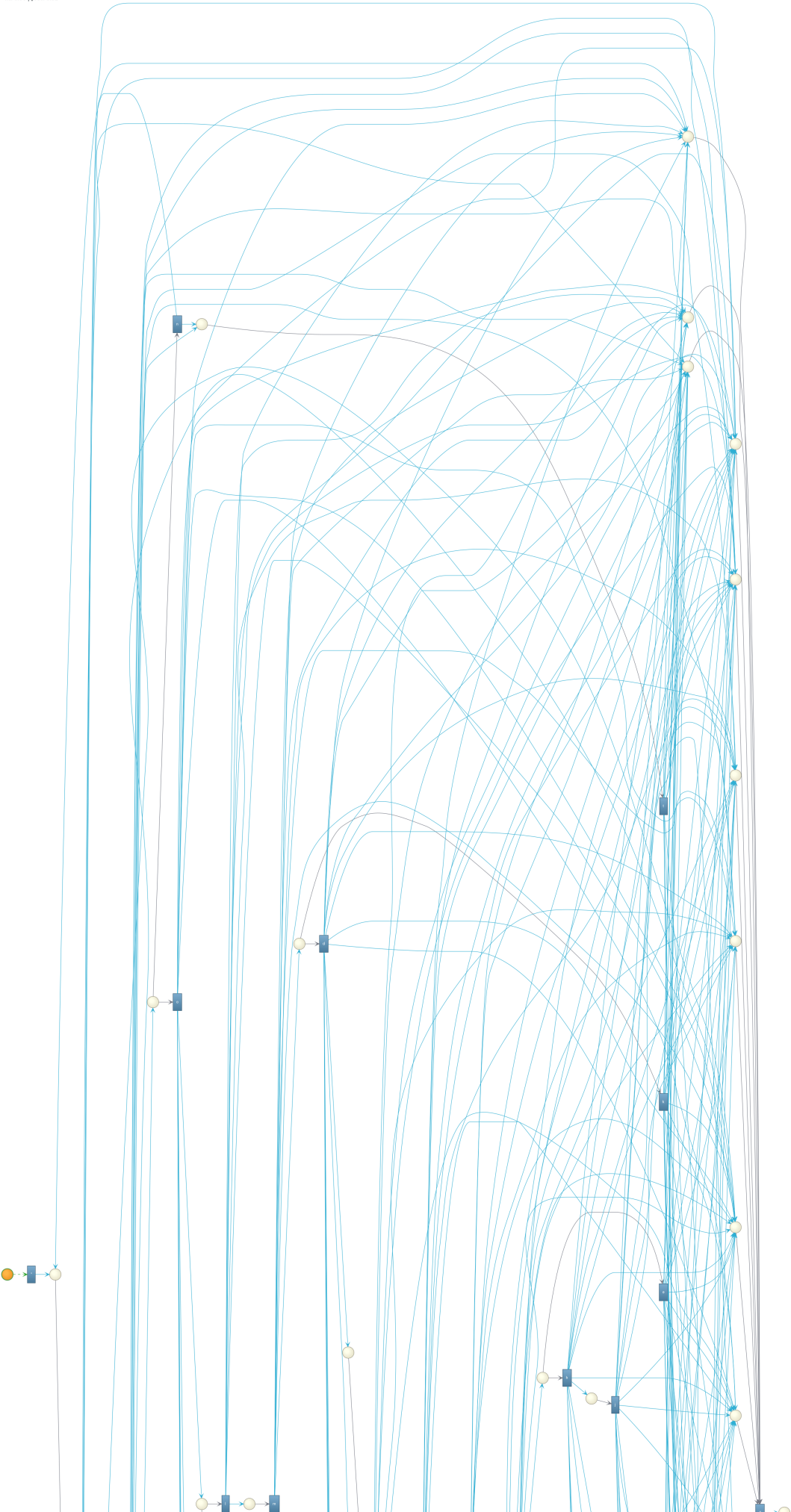
**2017.01**  
fit: 0.92 | prec: 0.12



2017.02  
fit: 0.98 | prec: 0.31

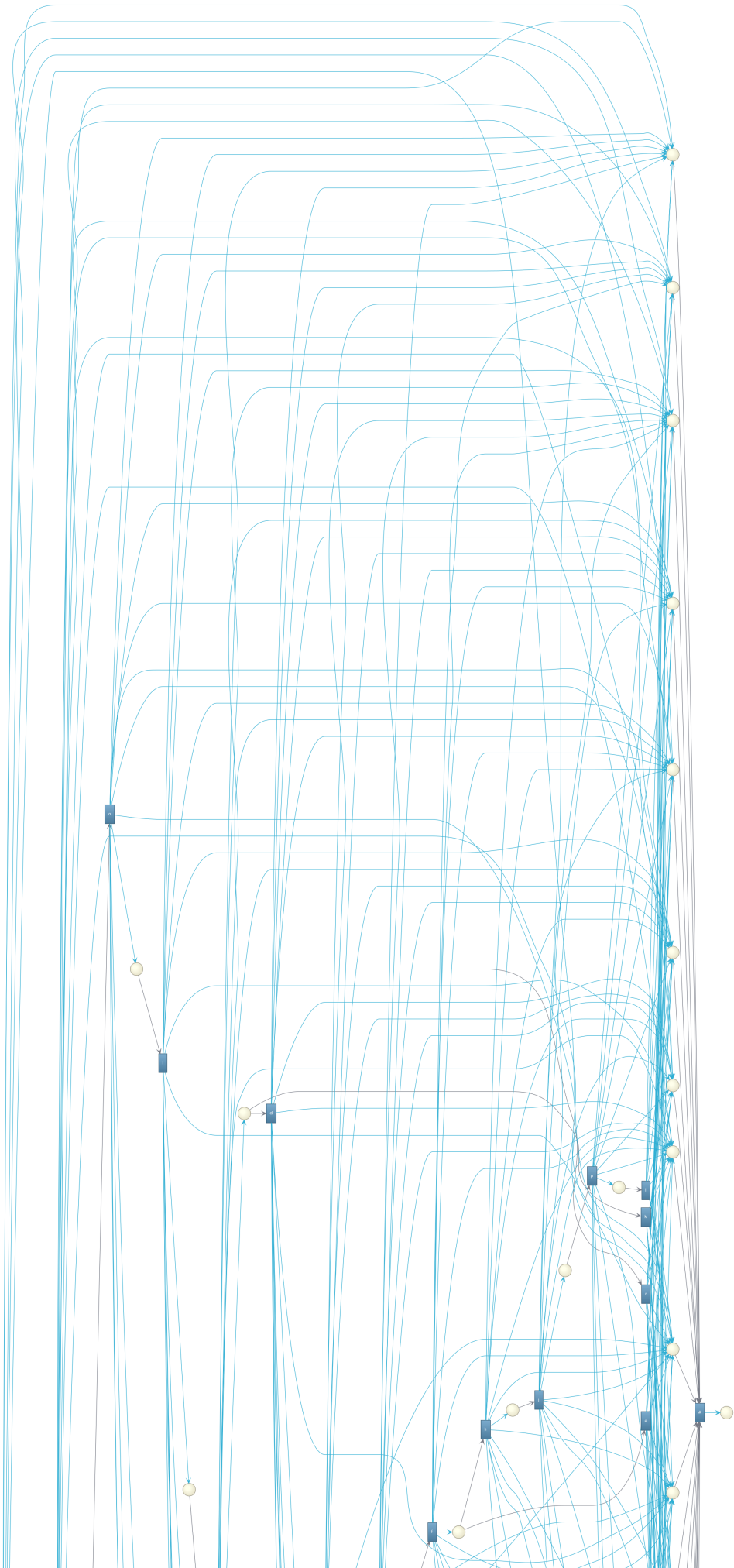


2017.02  
fit: 0.96 | prec: 0.32





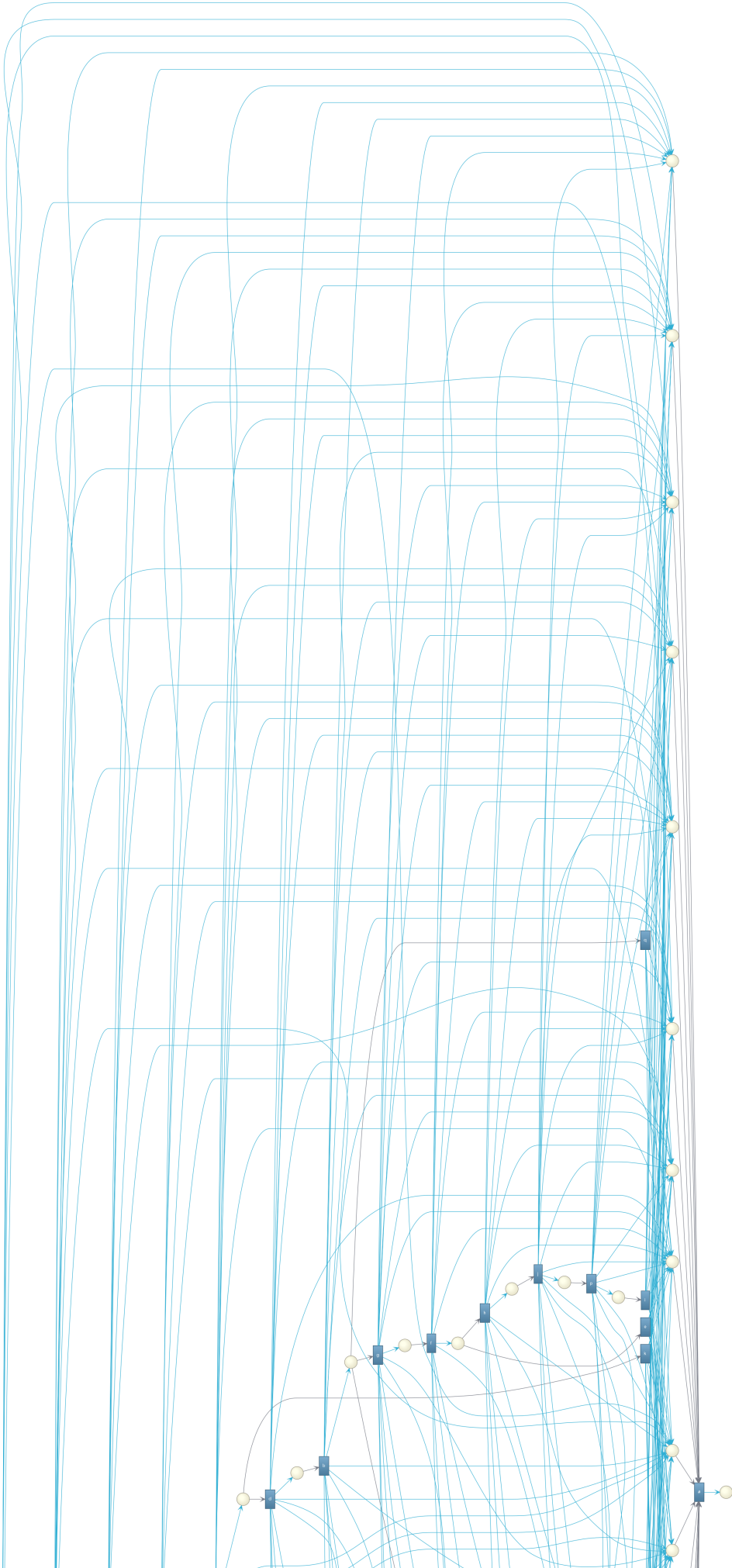
2017.02  
fit: 0.96 | prec: 0.32



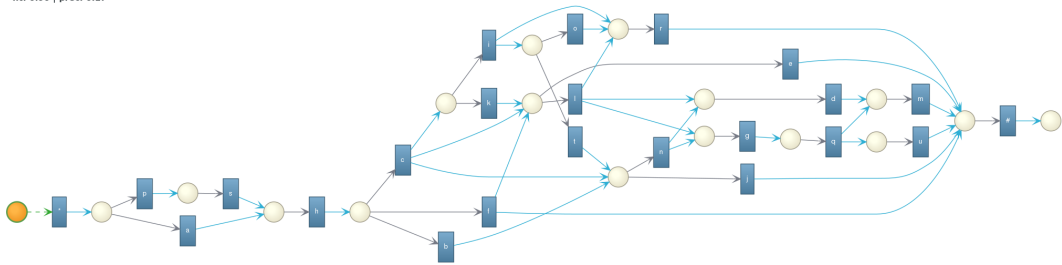


2017.02  
fit: 0.96 | prec: 0.32

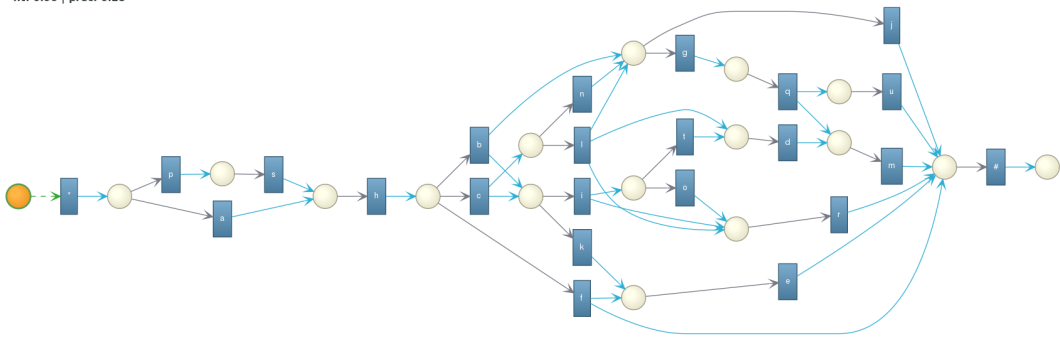
PUC-Rio - Certificação Digital Nº 1712662/CA



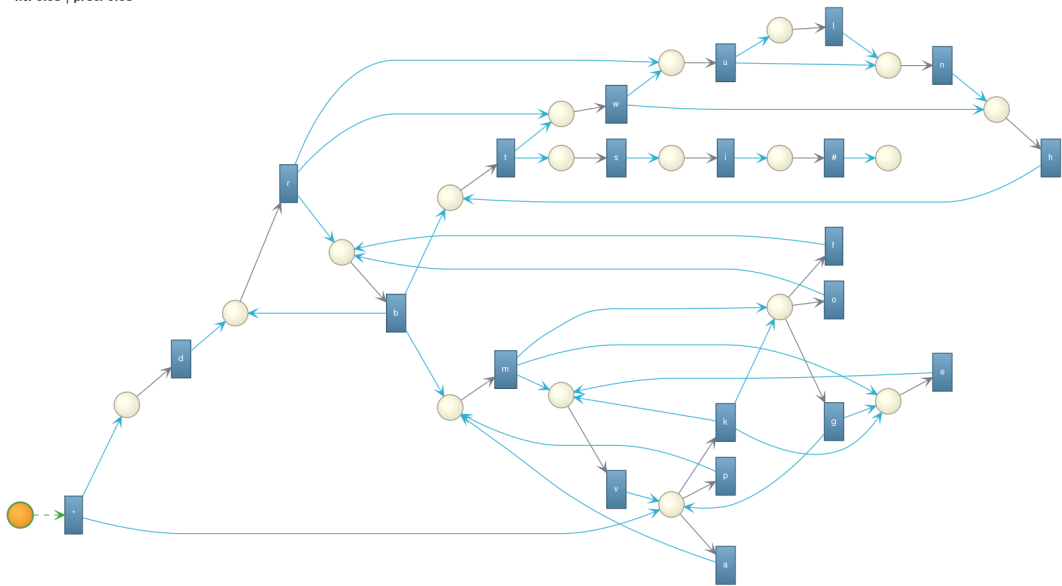
2017.03  
fit: 0.90 | prec: 0.17

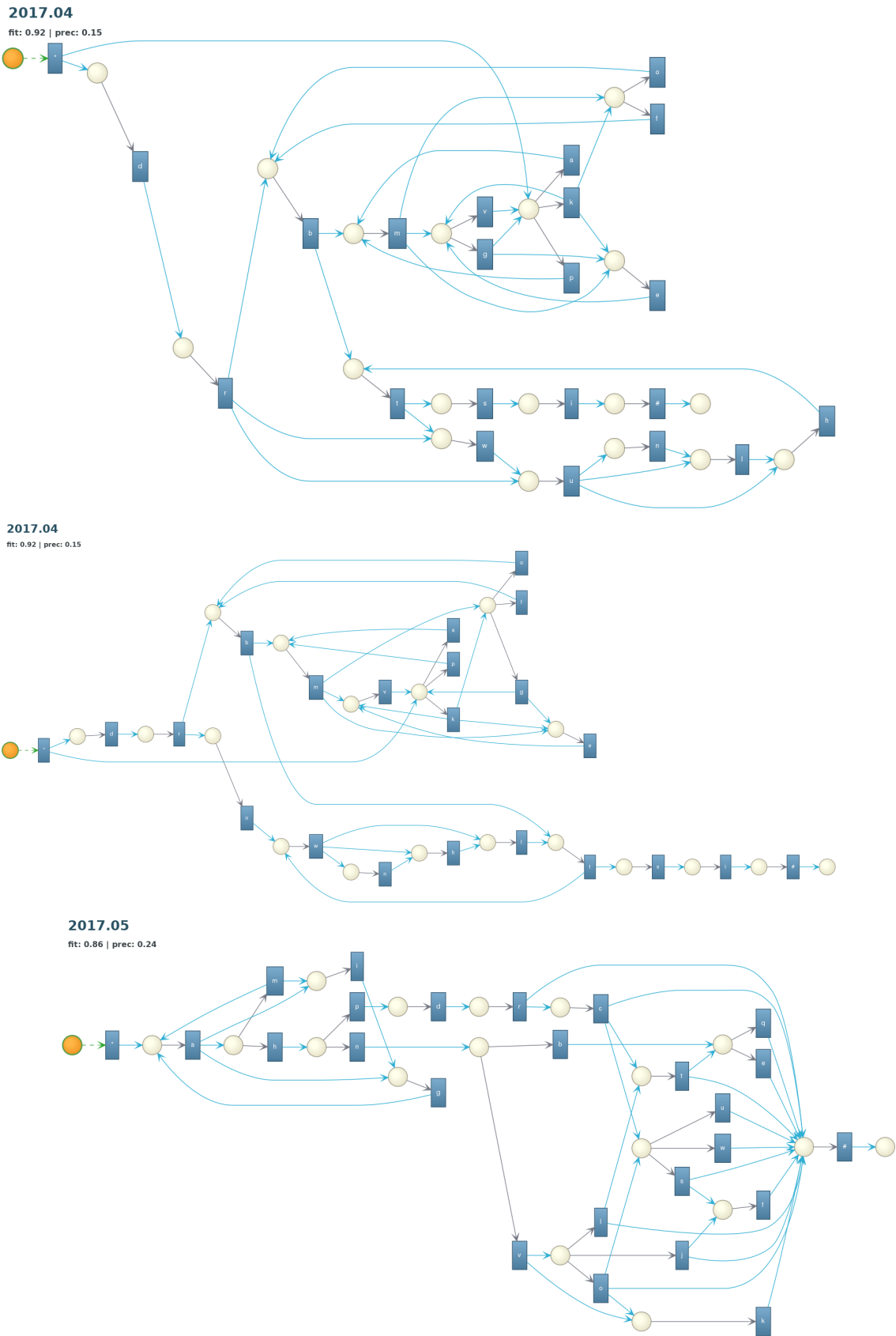


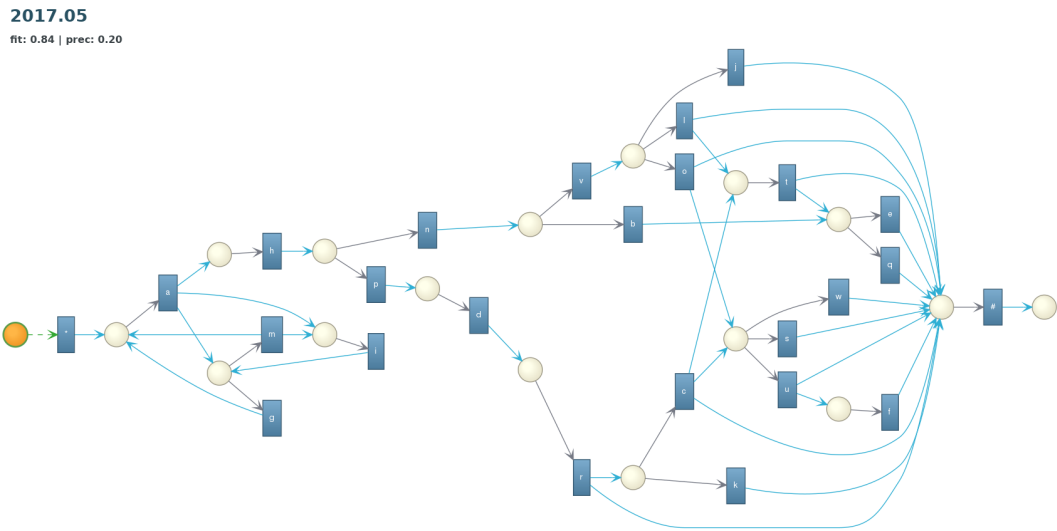
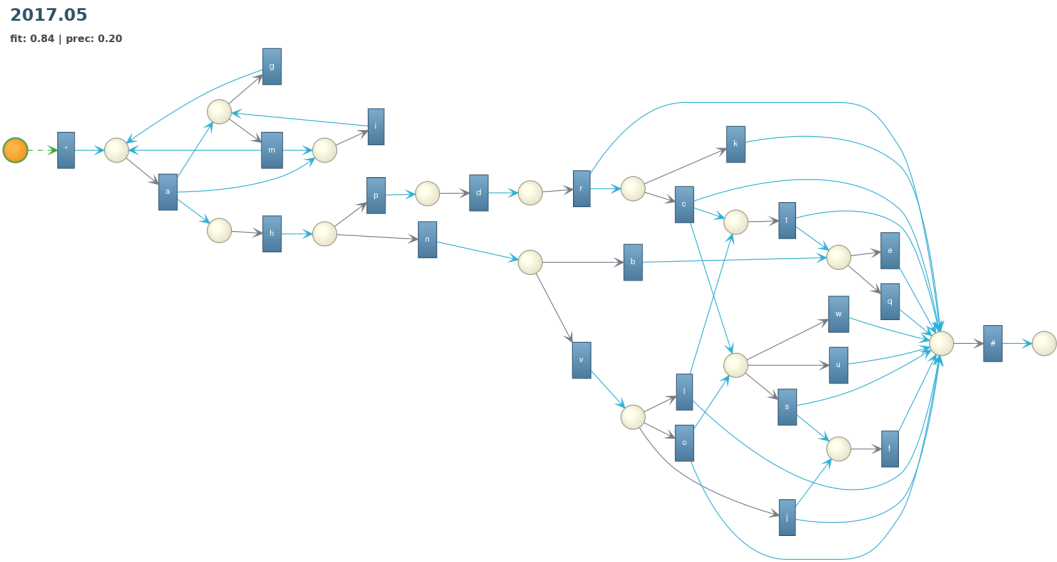
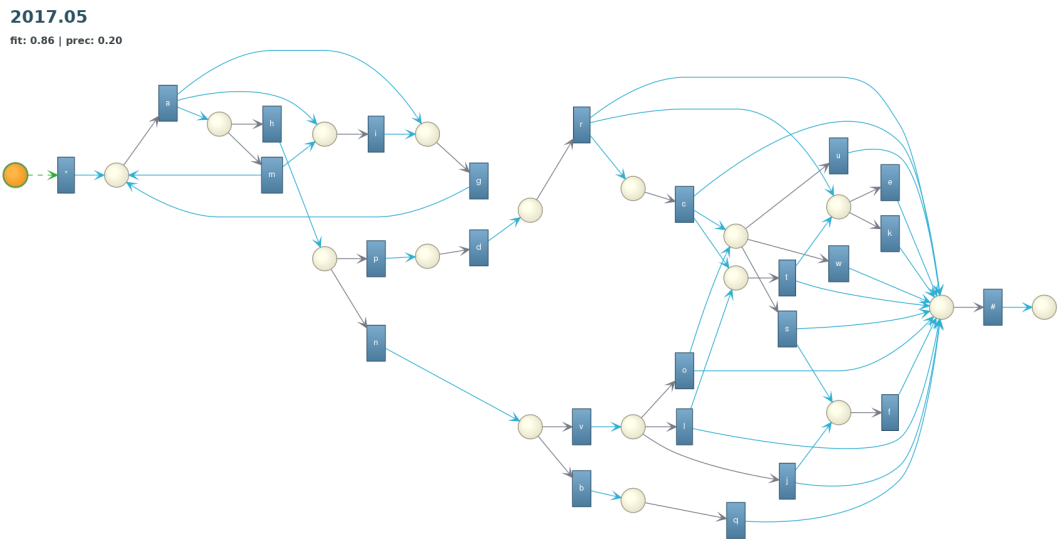
2017.03  
fit: 0.90 | prec: 0.28

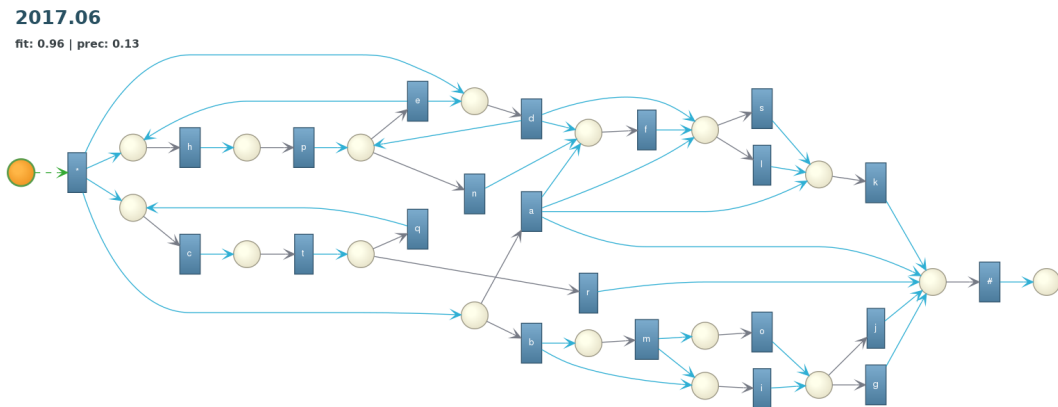
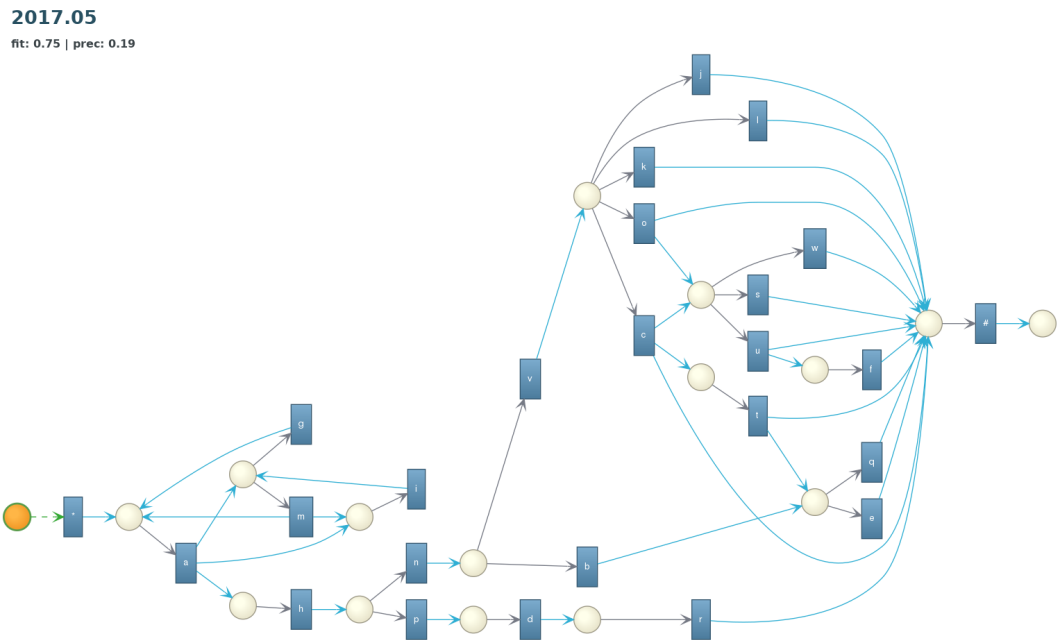
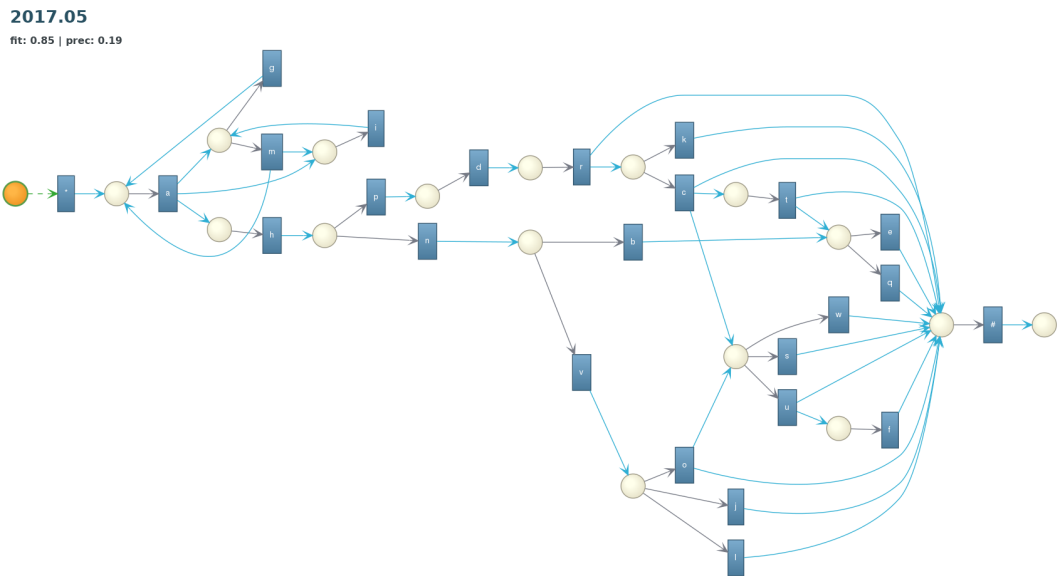


2017.04  
fit: 0.93 | prec: 0.08

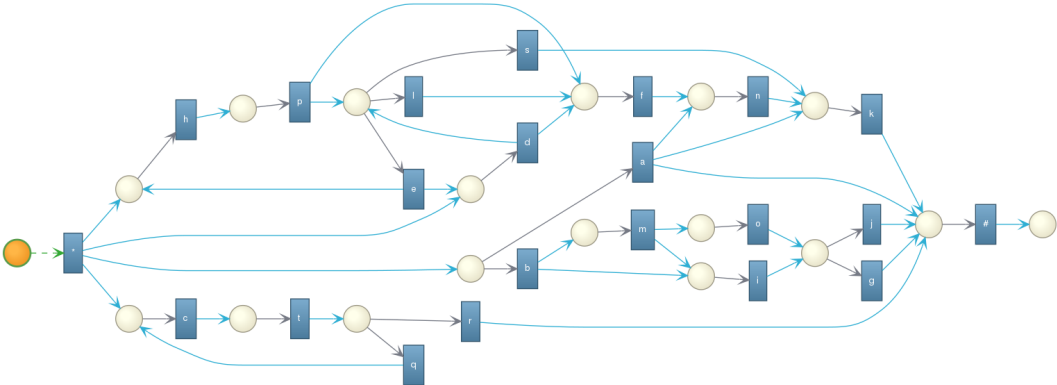




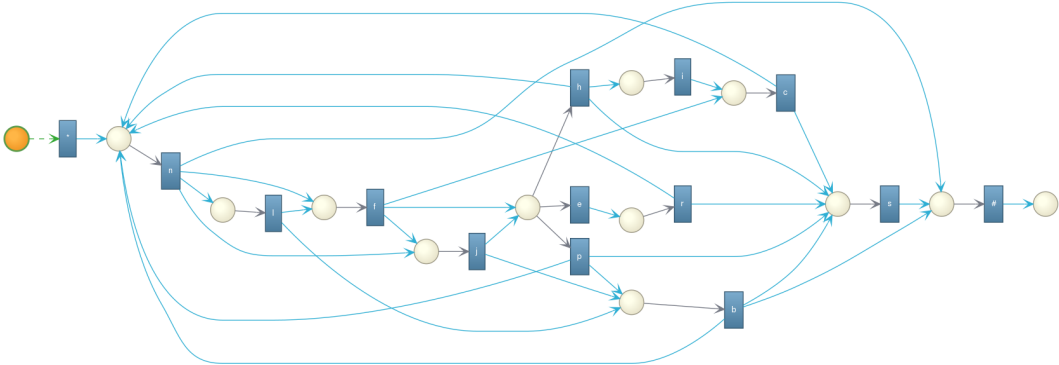




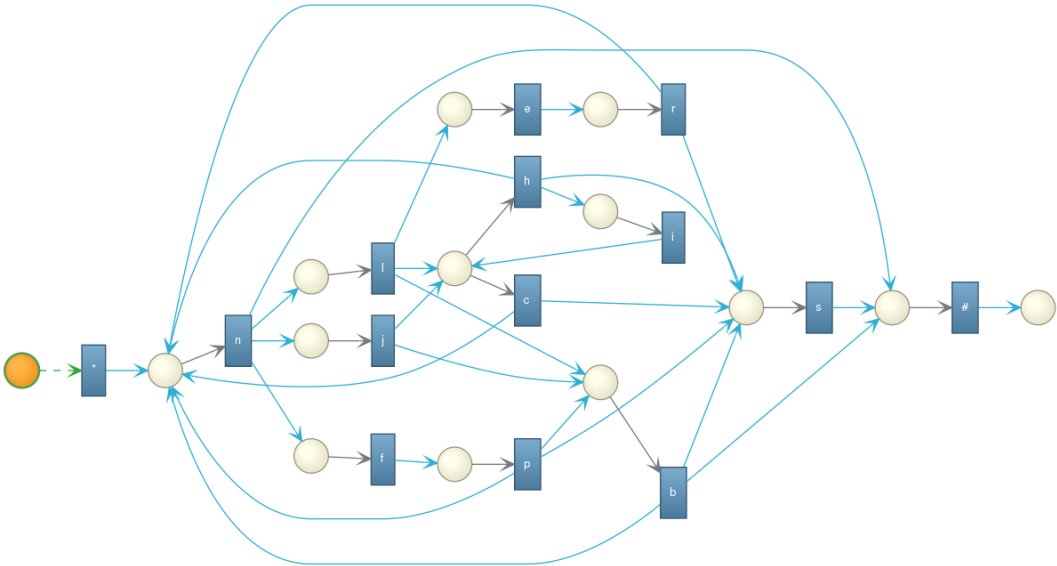
**2017.06**  
fit: 0.96 | prec: 0.12



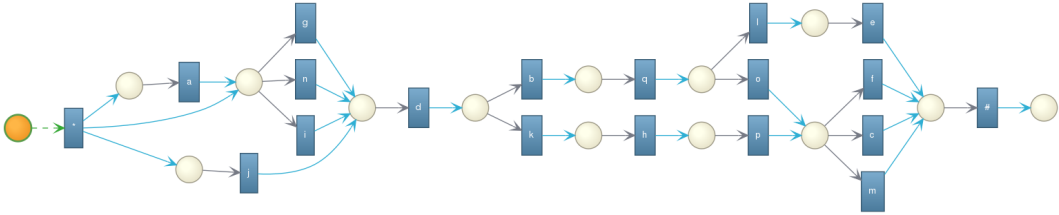
**2017.07**  
fit: 0.97 | prec: 0.16

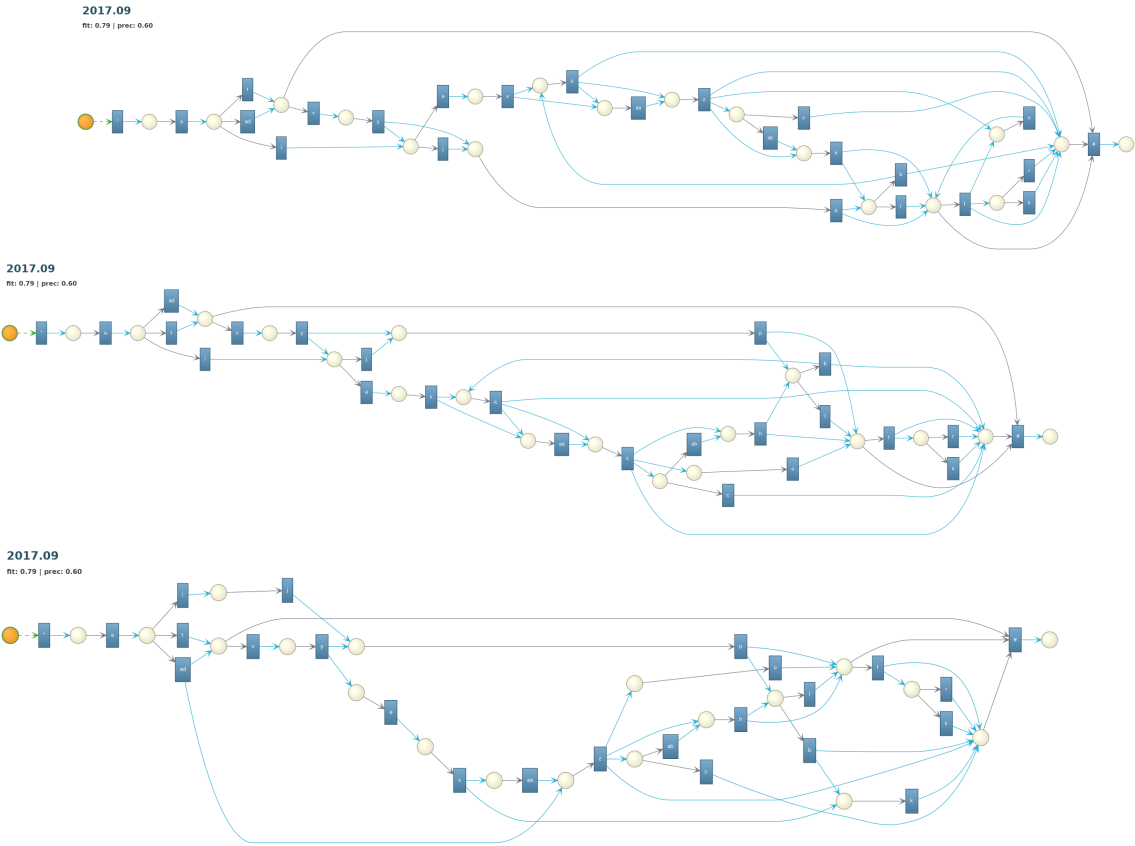


**2017.07**  
fit: 0.97 | prec: 0.18

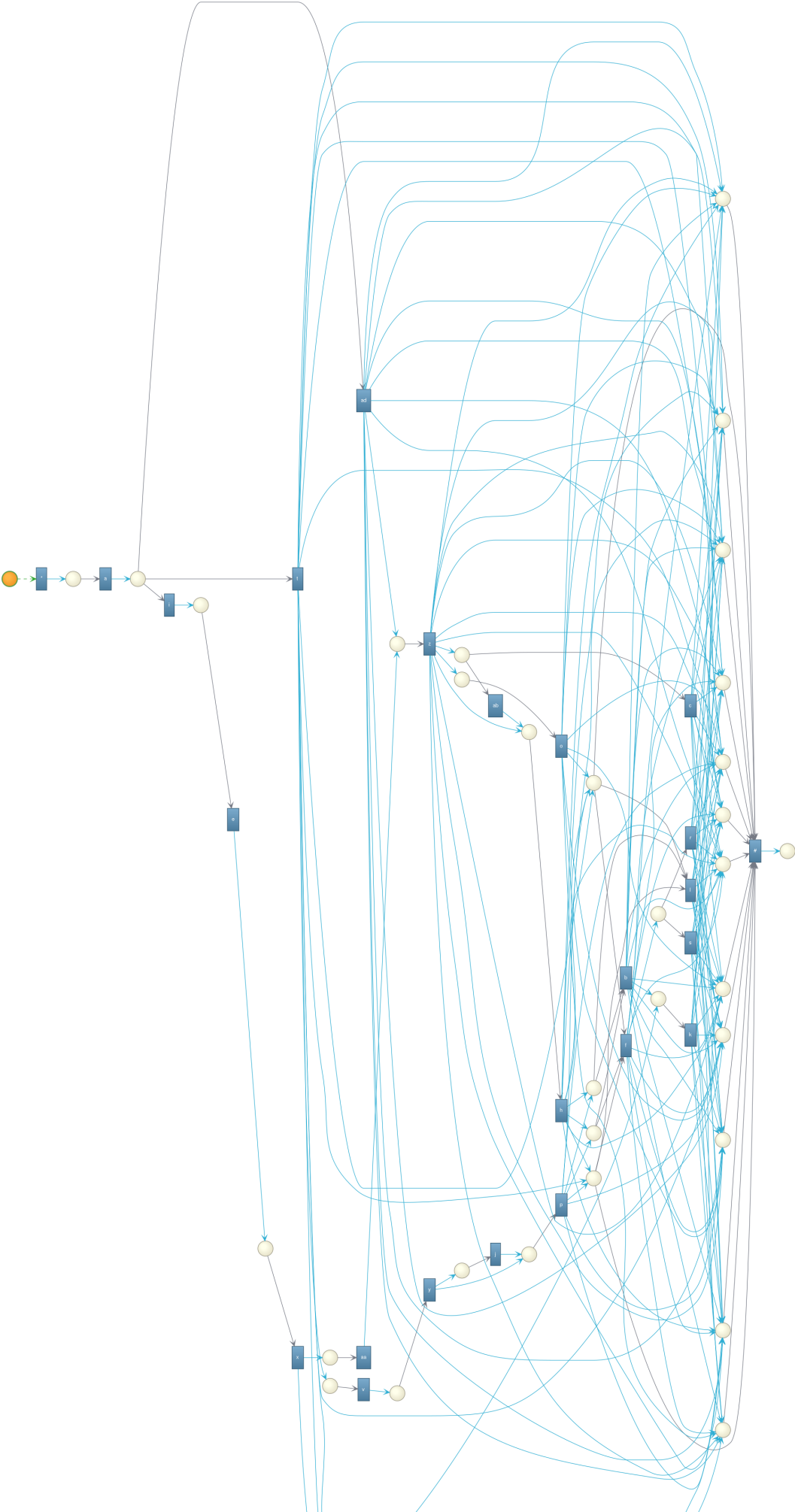


**2017.08**  
fit: 0.89 | prec: 0.23



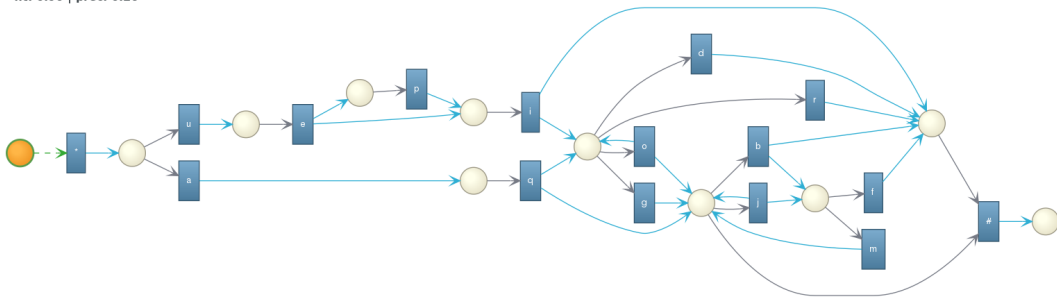


2017.09  
fit: 0.83 | prec: 0.67

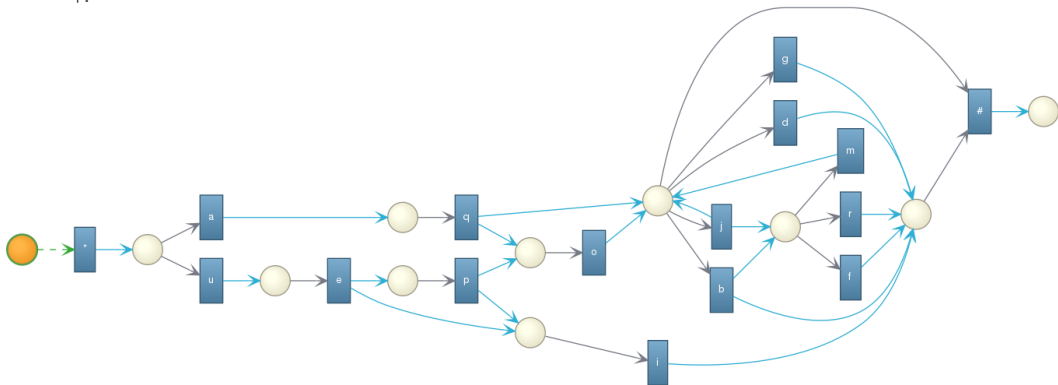




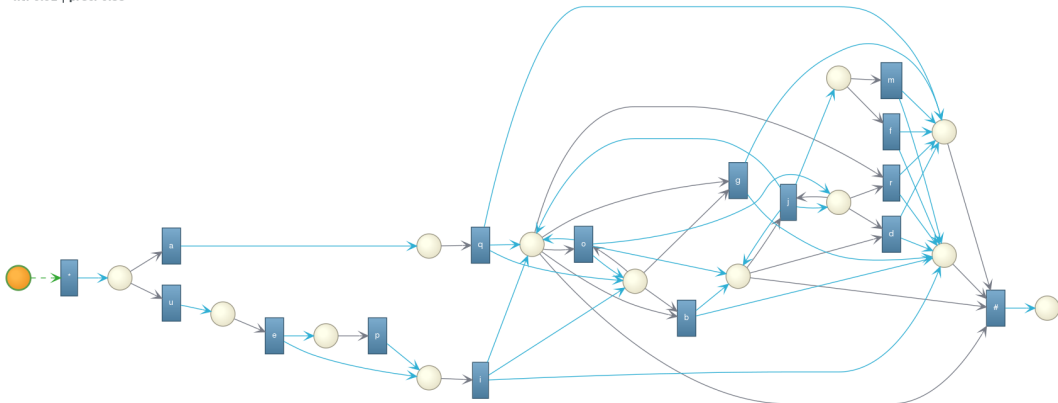
2017.10  
fit: 0.90 | prec: 0.26



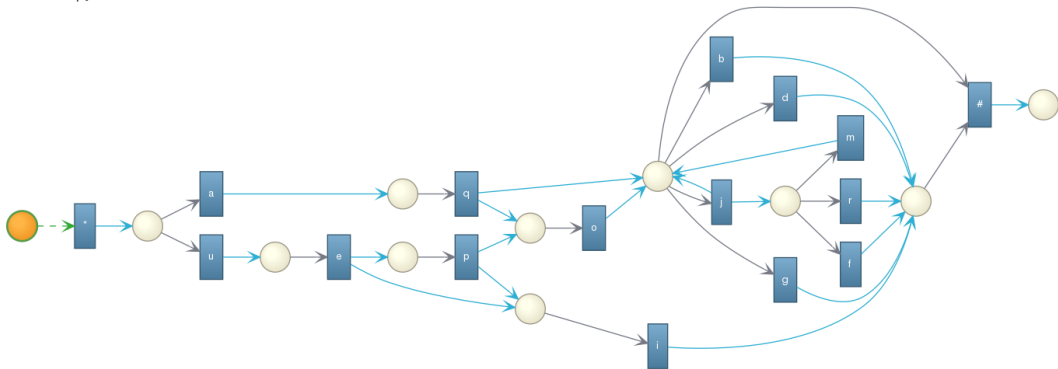
2017.10  
fit: 0.90 | prec: 0.42



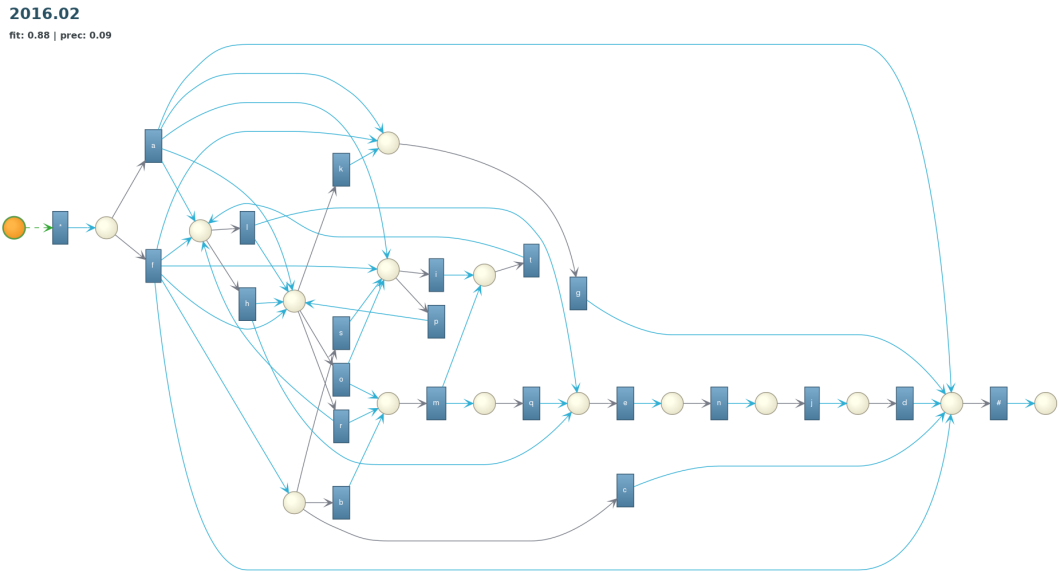
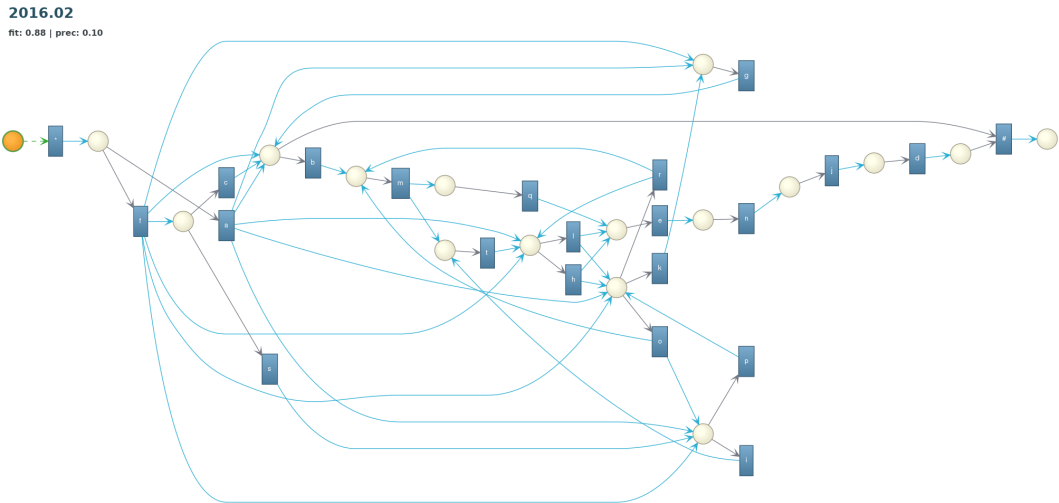
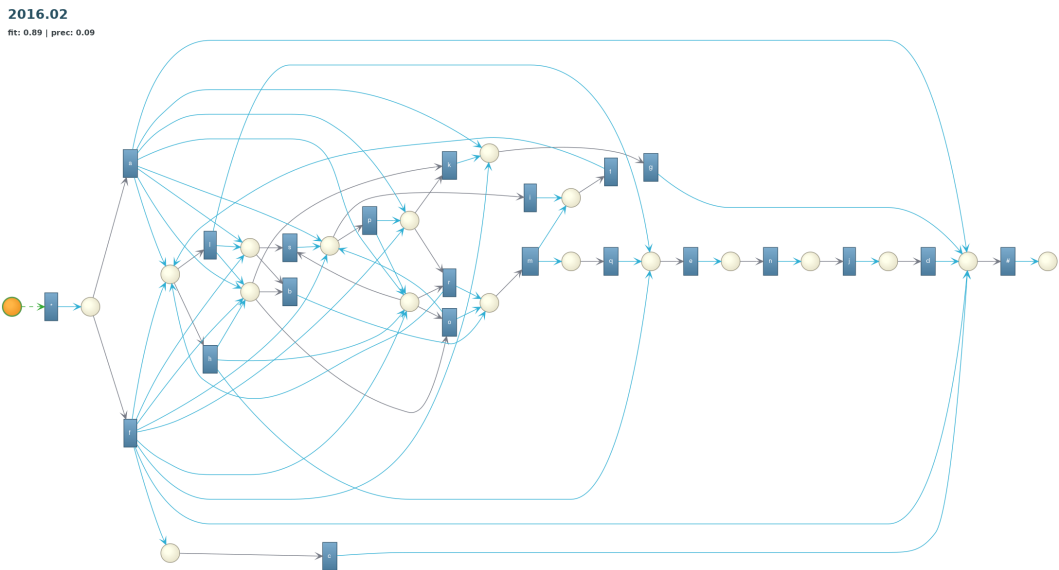
2017.10  
fit: 0.92 | prec: 0.33



2017.10  
fit: 0.91 | prec: 0.42

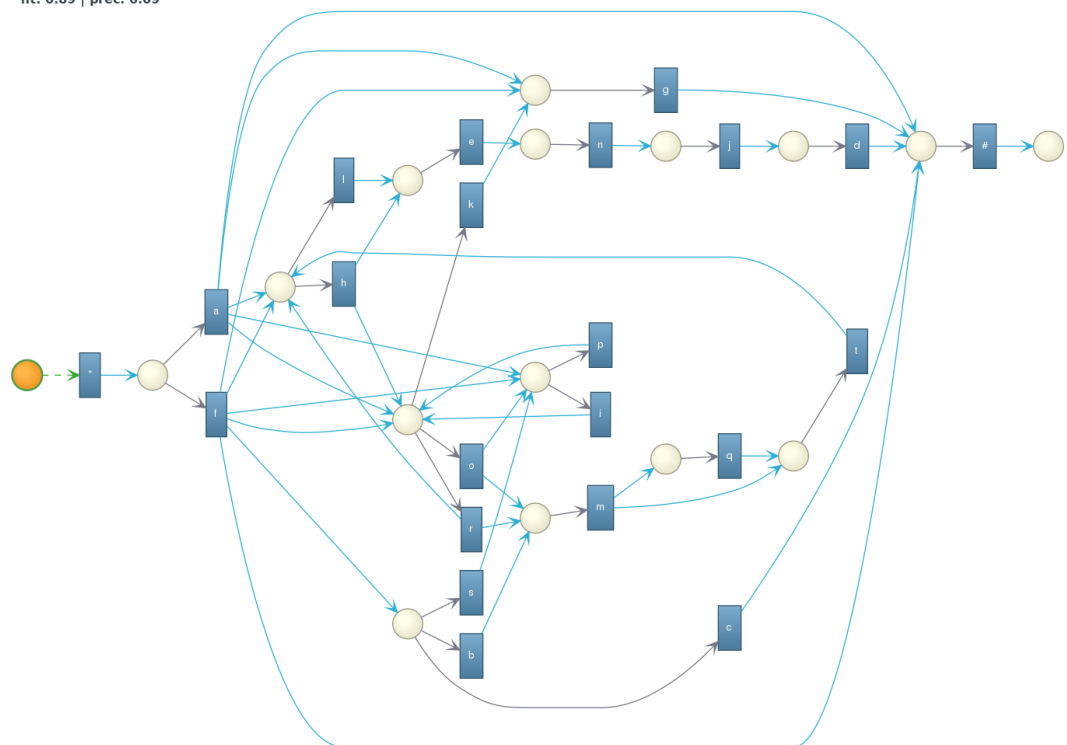






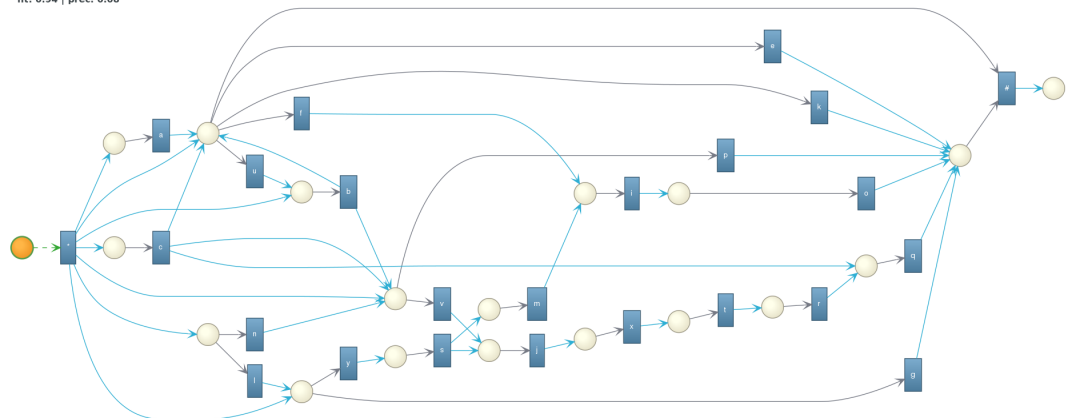
2016.02

fit: 0.89 | prec: 0.09



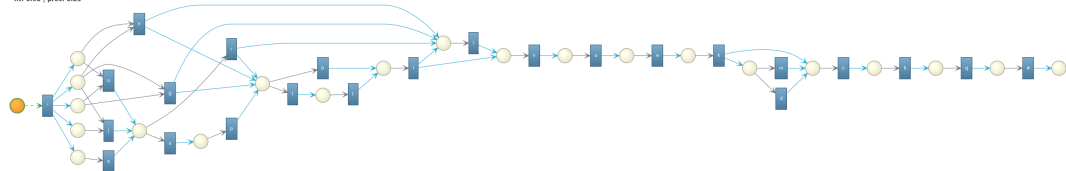
2016.03

fit: 0.94 | prec: 0.08



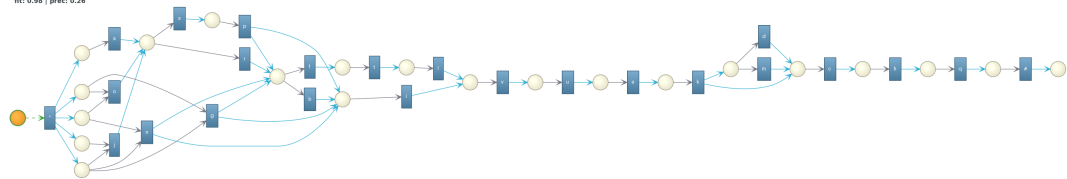
2016.04

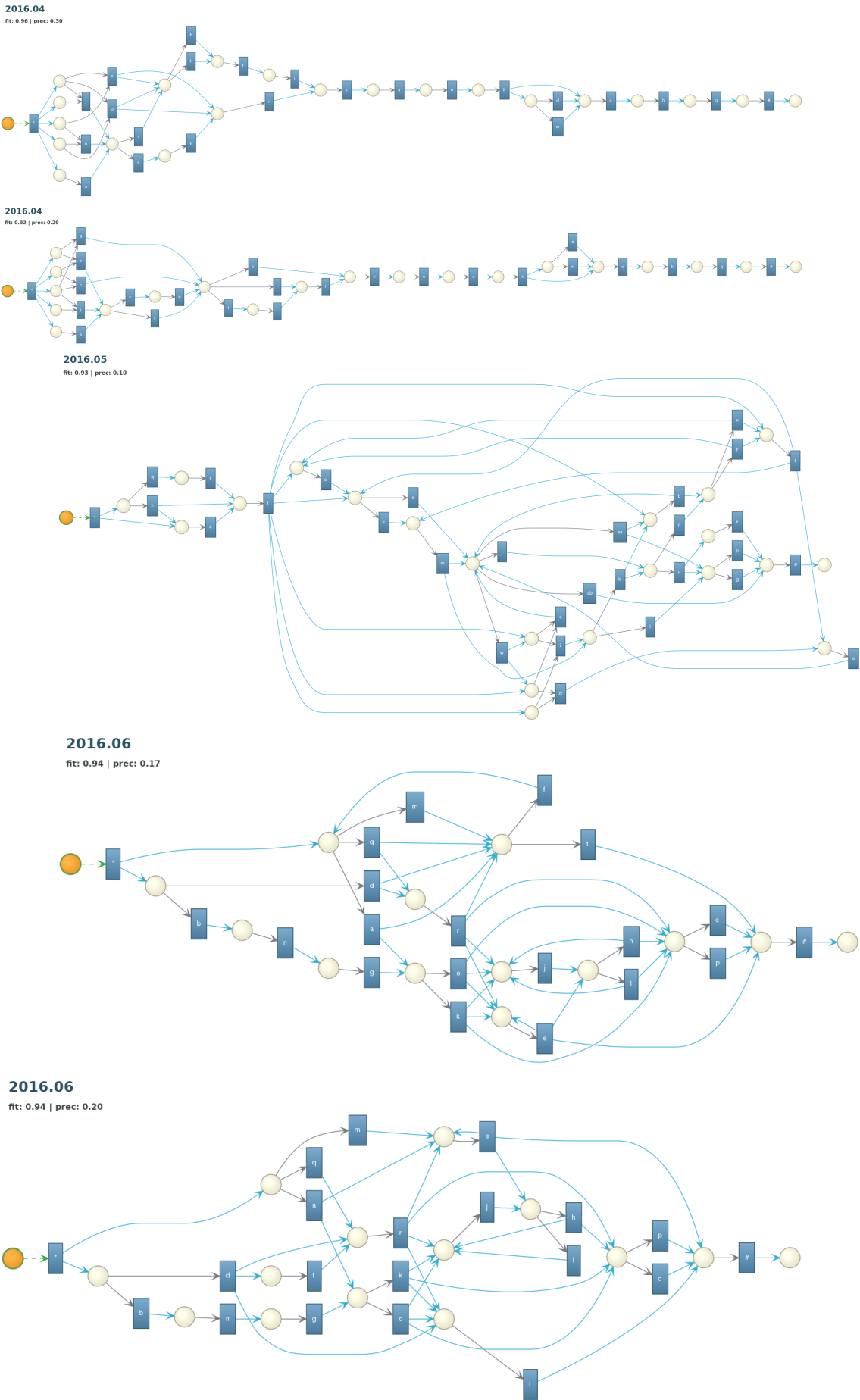
fit: 0.98 | prec: 0.25



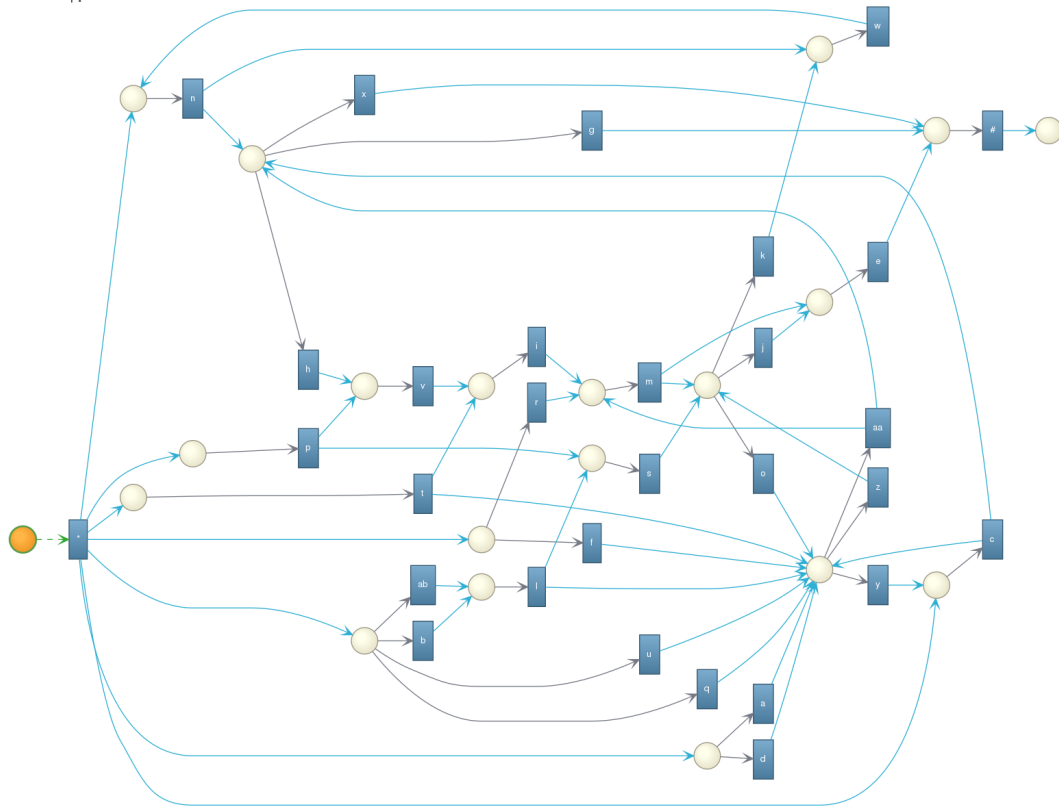
2016.04

fit: 0.98 | prec: 0.26

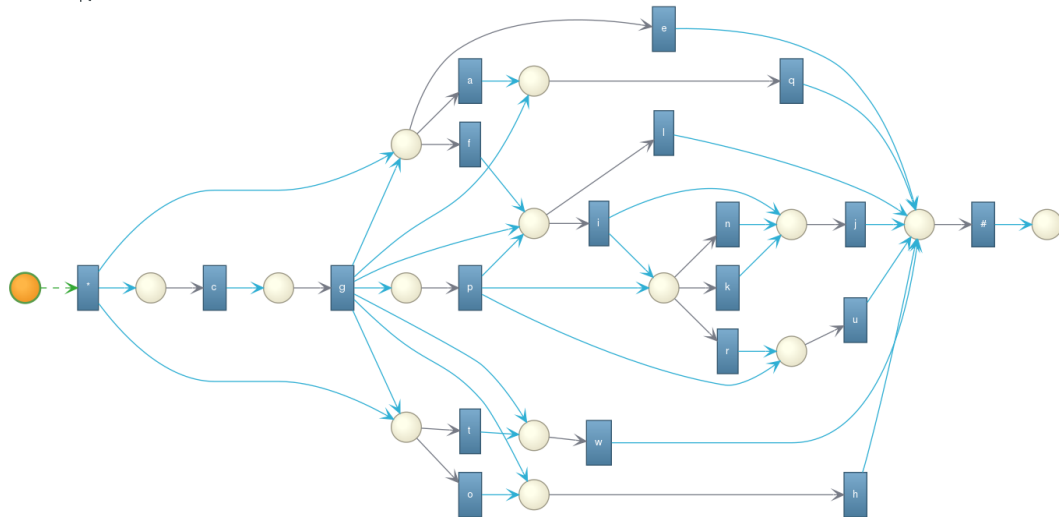




fit: 0.92 | prec: 0.08

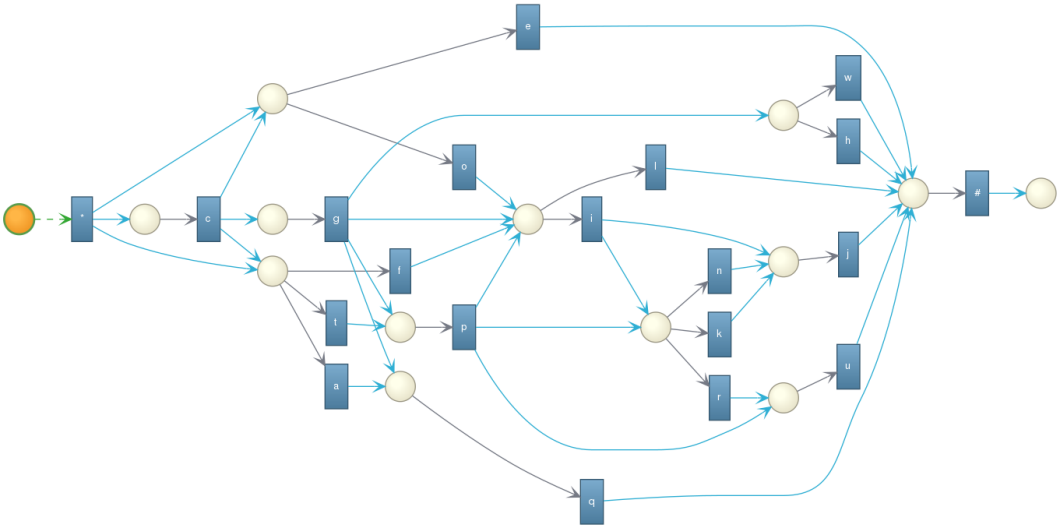


fit: 0.95 | prec: 0.09



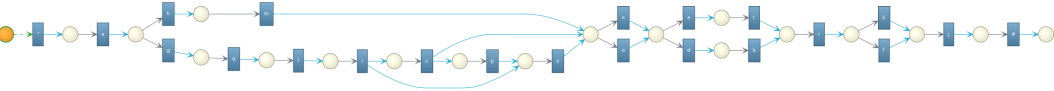
2016.08

fit: 0.96 | prec: 0.09



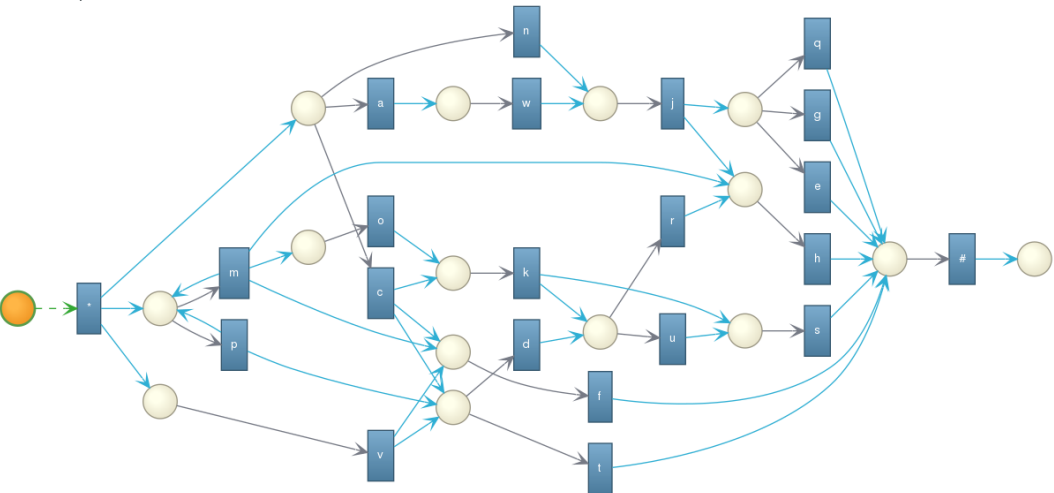
2016.09

fit: 0.92 | prec: 0.70

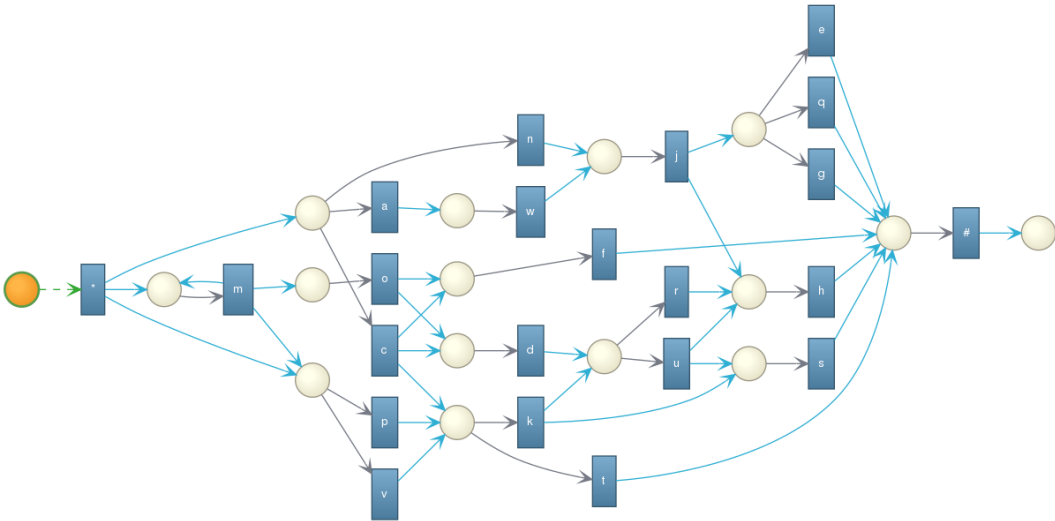


2016.10

fit: 0.91 | prec: 0.11



2016.10  
fit: 0.91 | prec: 0.12

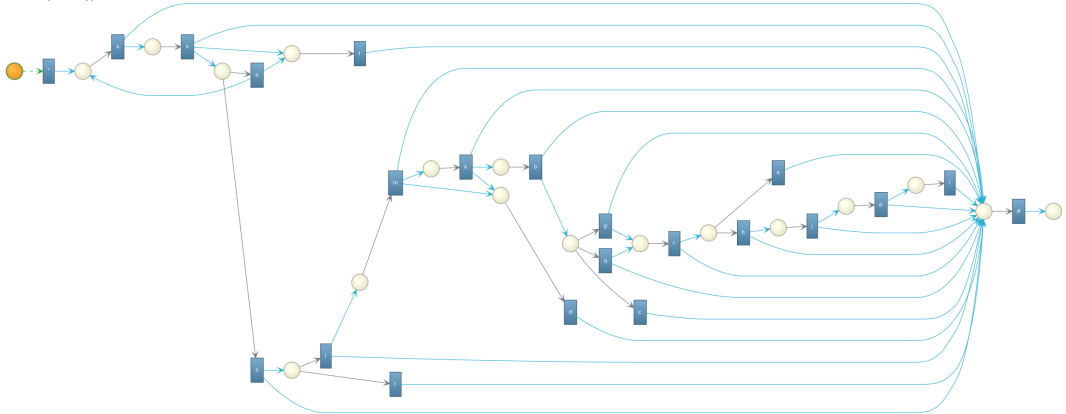




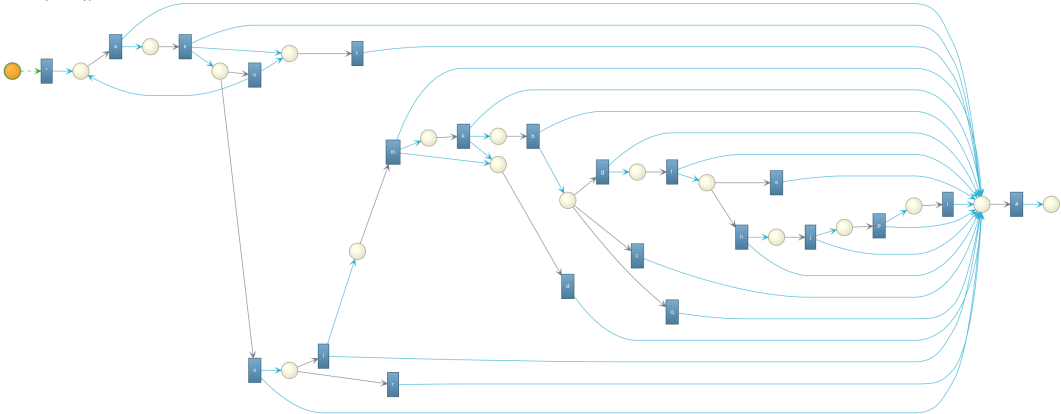
D

## Process models generated using the Risk-Prone model

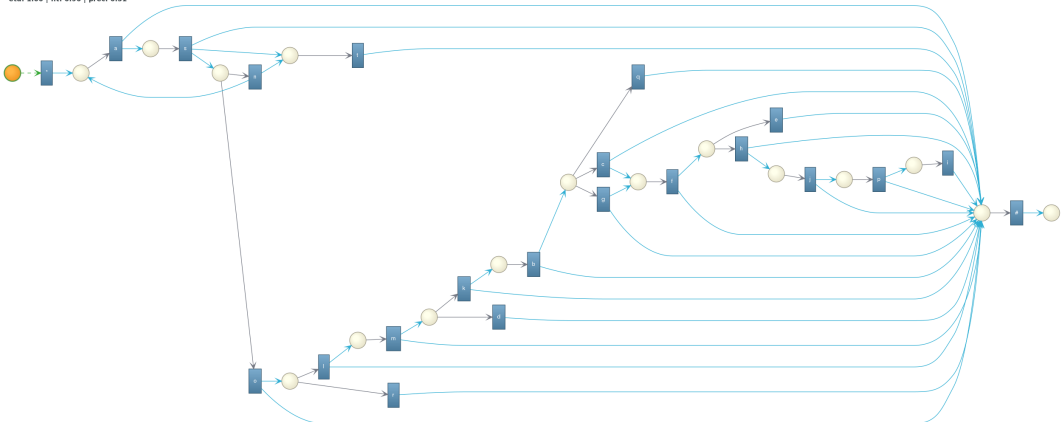
2017.02  
eta: 1.00 | fti: 0.97 | prec: 0.27

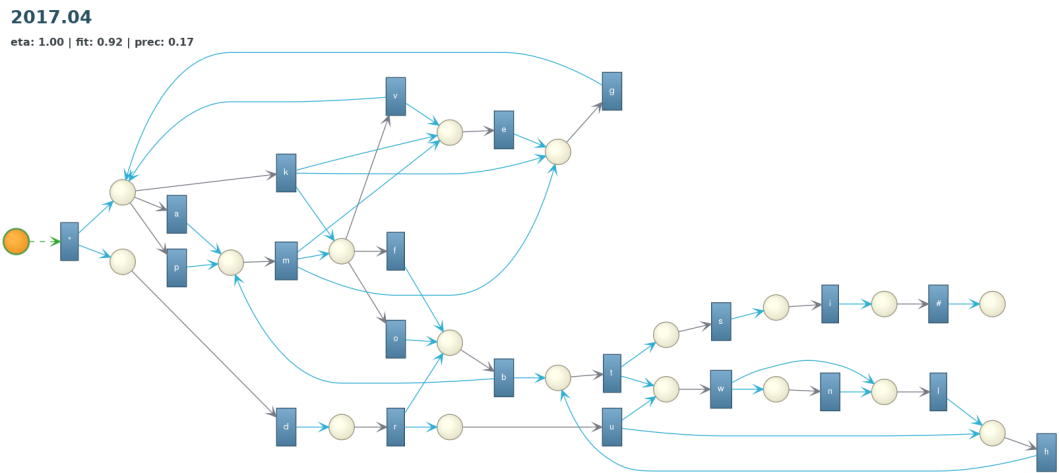
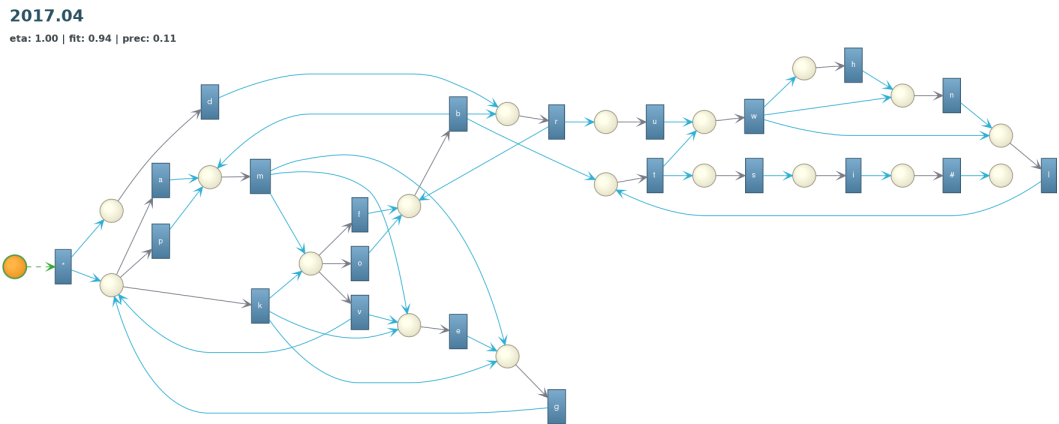
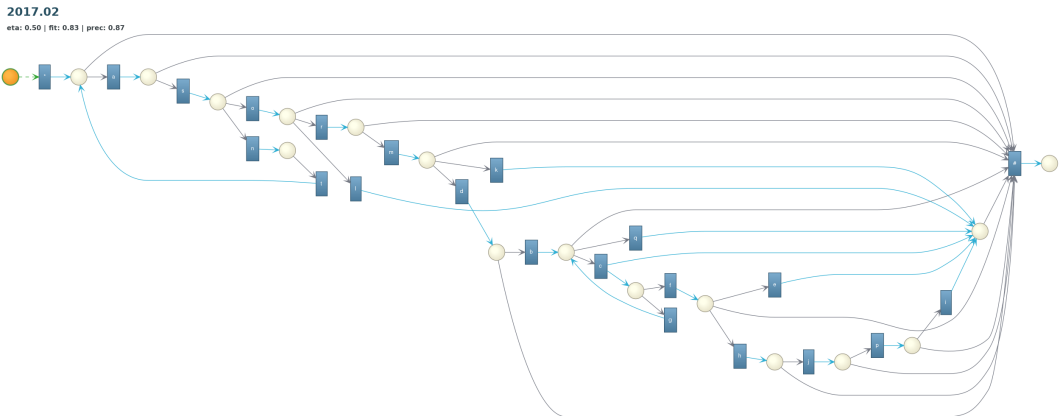
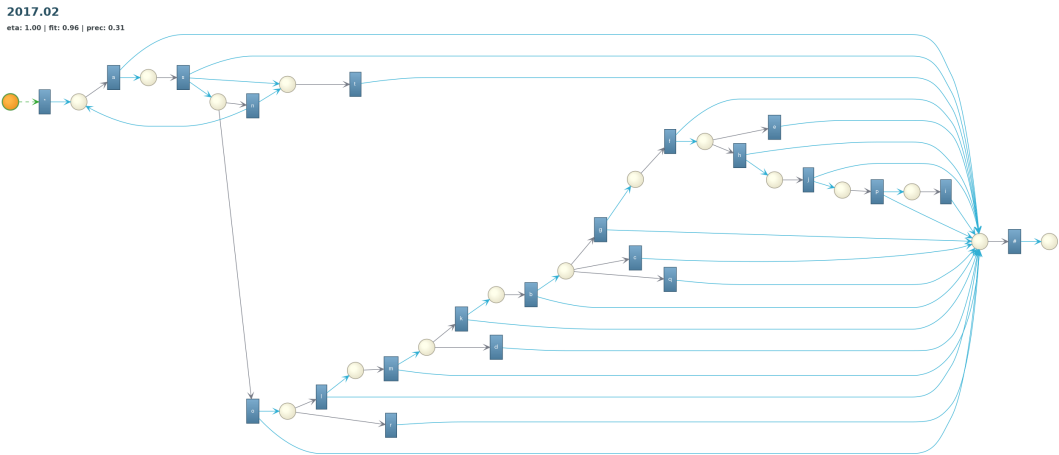


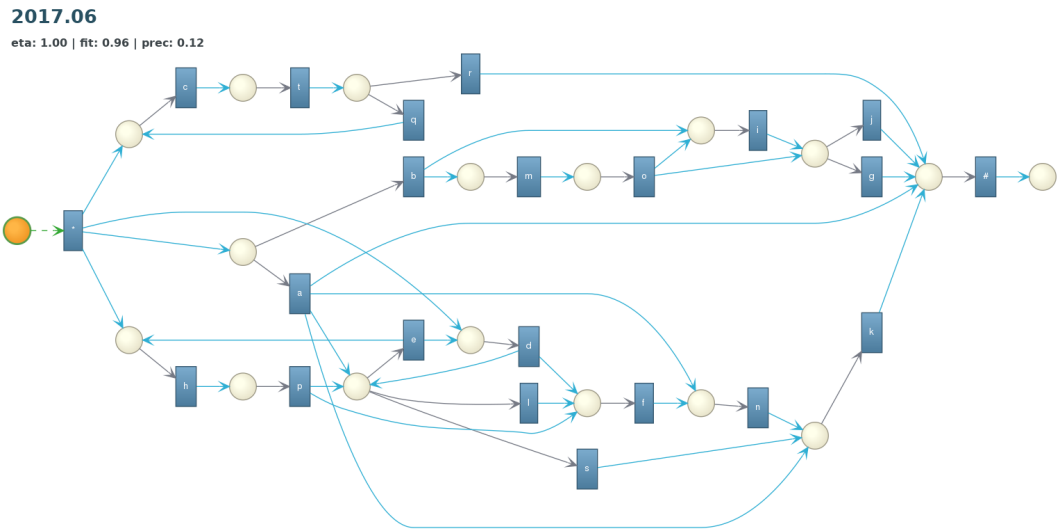
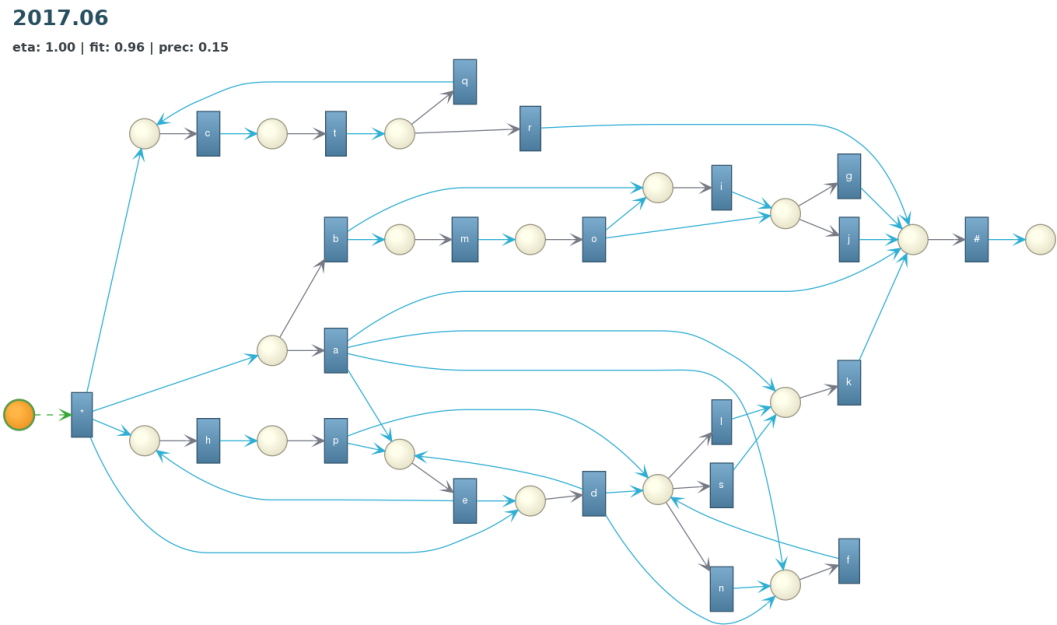
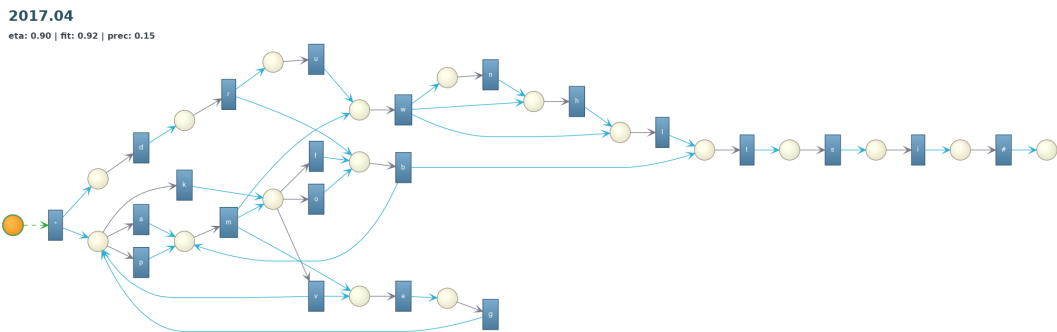
2017.02  
eta: 1.00 | fti: 0.97 | prec: 0.30

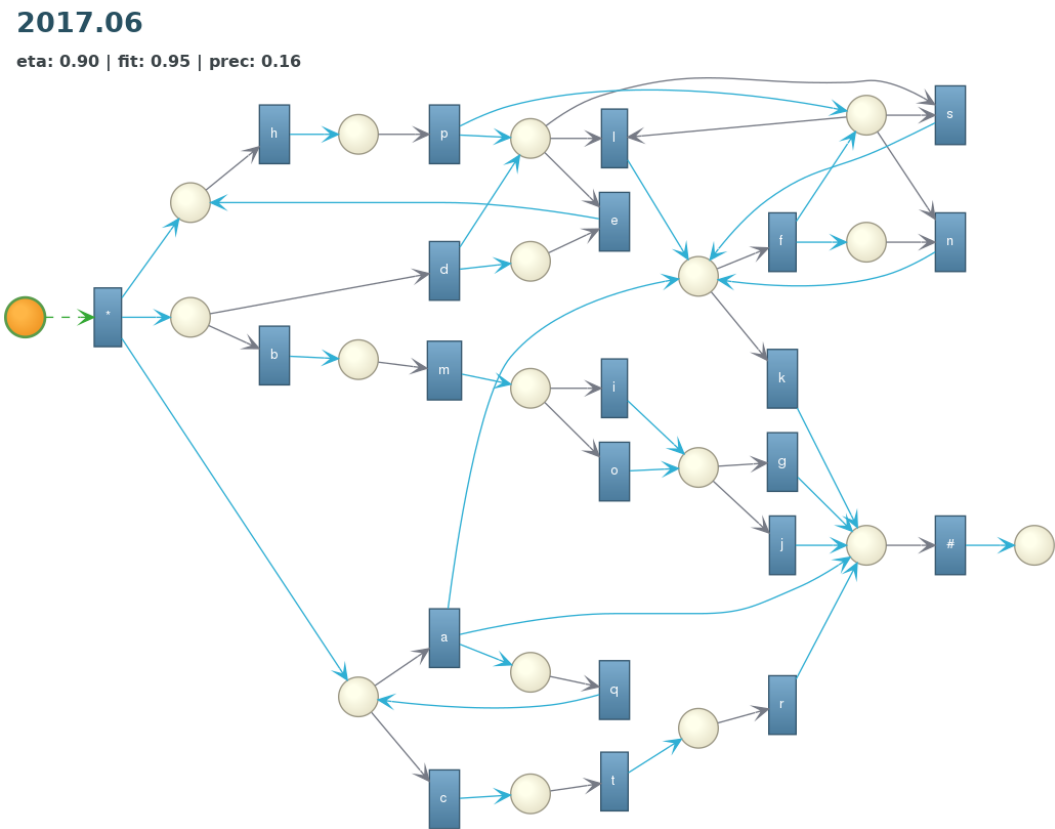
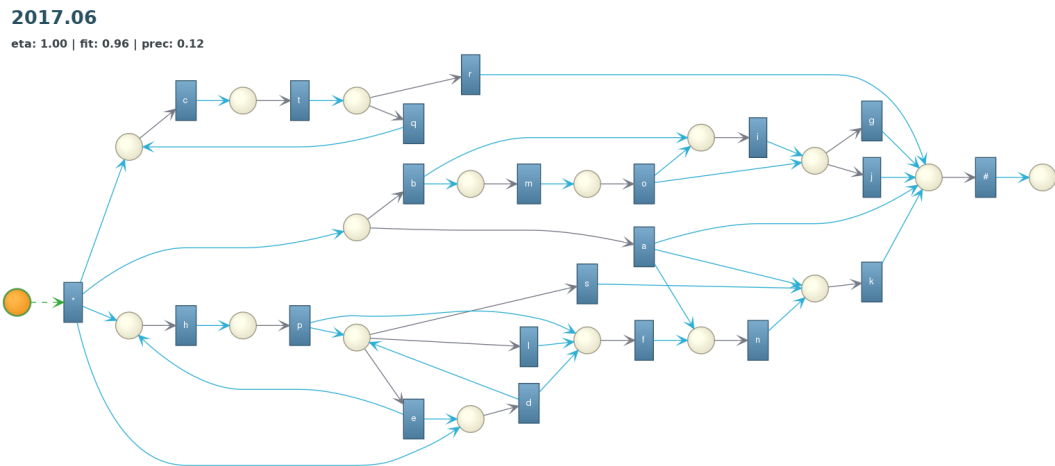


2017.02  
eta: 1.00 | fti: 0.96 | prec: 0.31



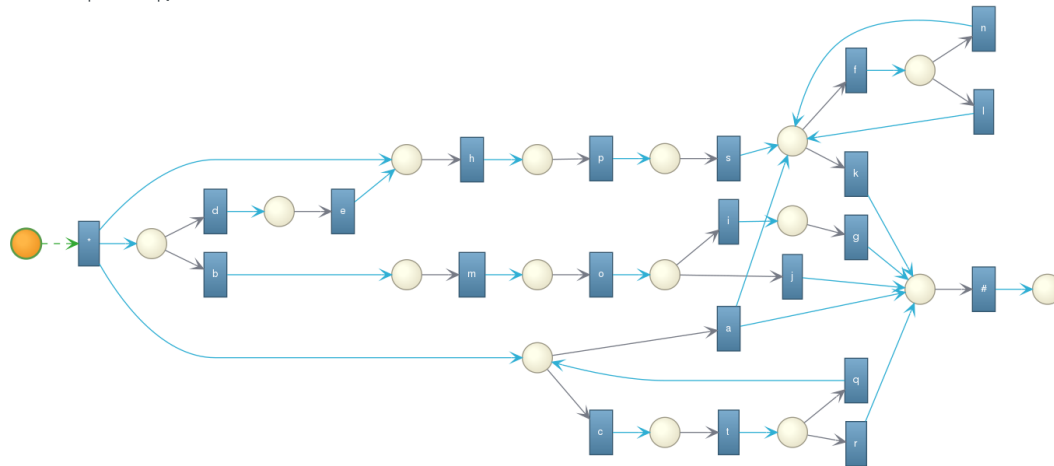




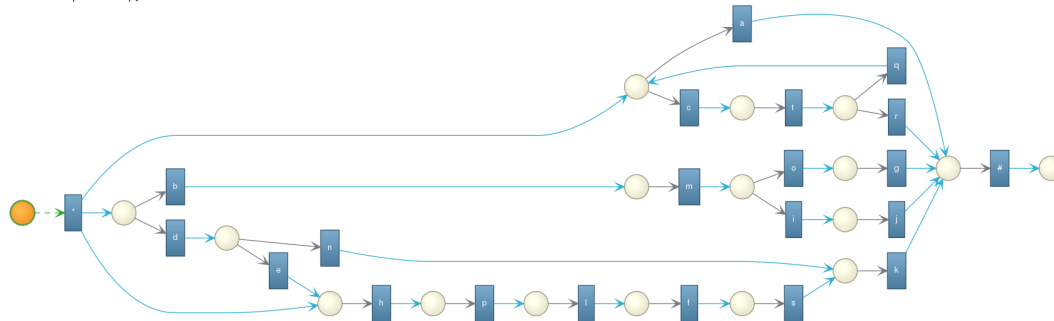


**2017.06**

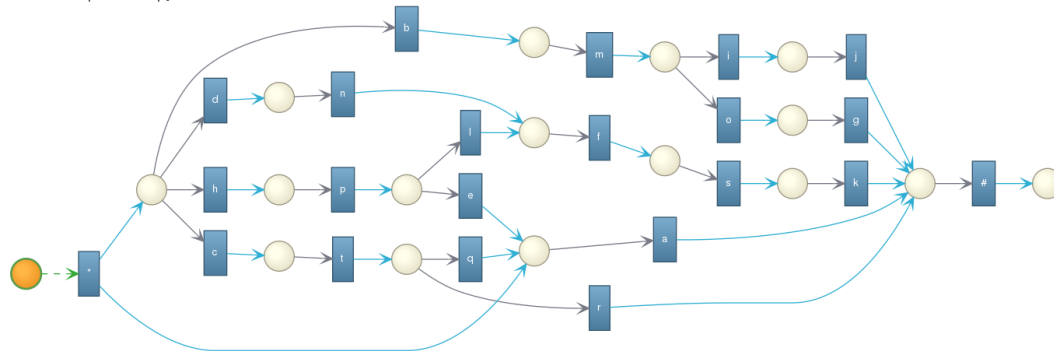
eta: 0.90 | fit: 0.96 | prec: 0.25

**2017.06**

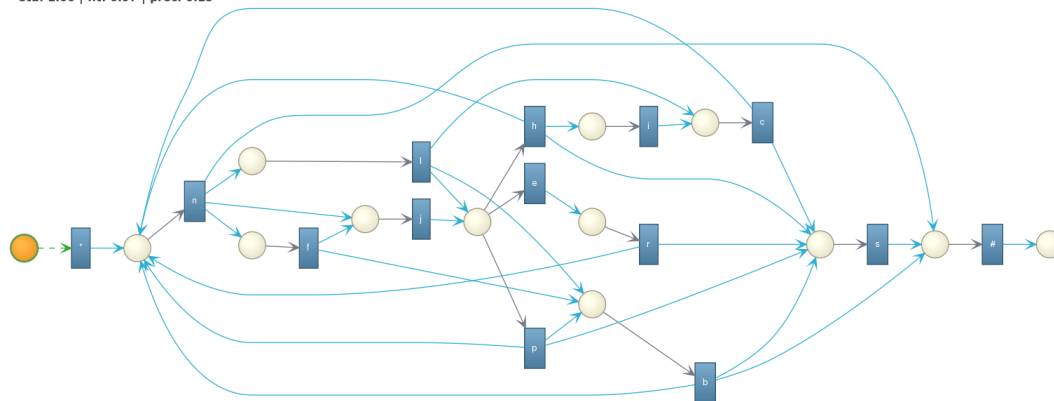
eta: 0.80 | fit: 0.95 | prec: 0.37

**2017.06**

eta: 0.60 | fit: 0.87 | prec: 0.70

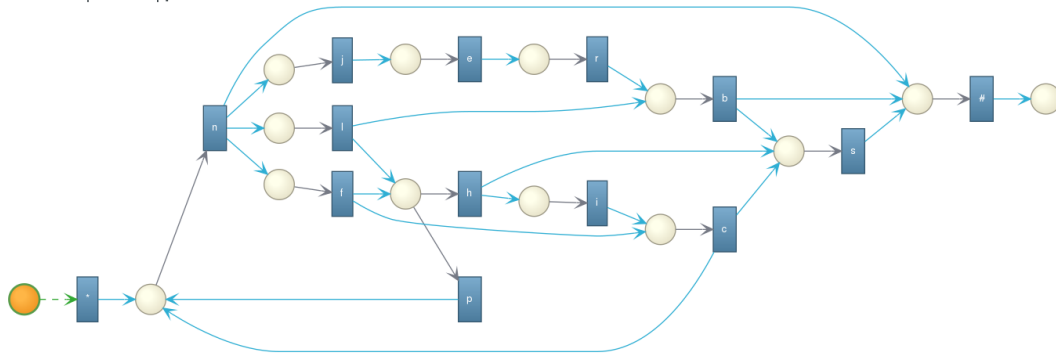
**2017.07**

eta: 1.00 | fit: 0.97 | prec: 0.15

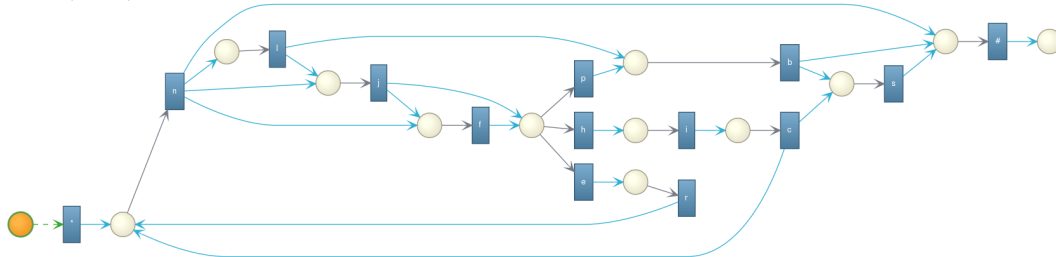


**2017.07**

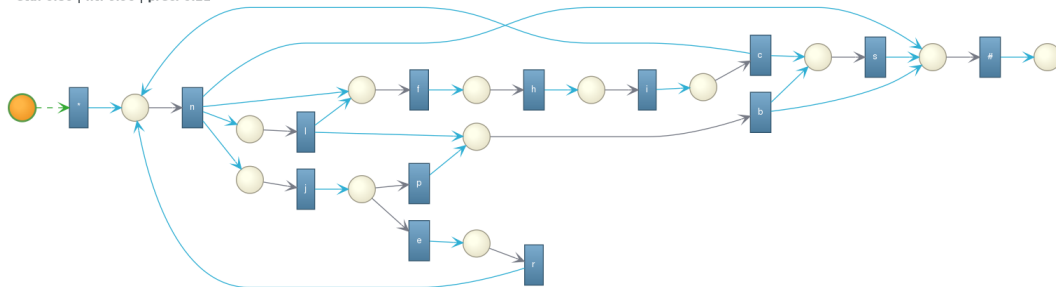
eta: 0.90 | fit: 0.97 | prec: 0.18

**2017.07**

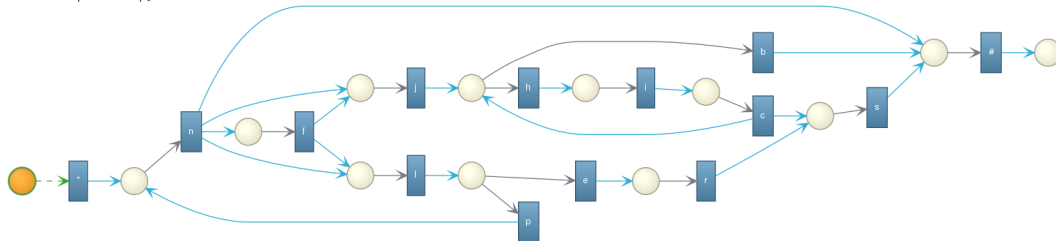
eta: 0.80 | fit: 0.96 | prec: 0.20

**2017.07**

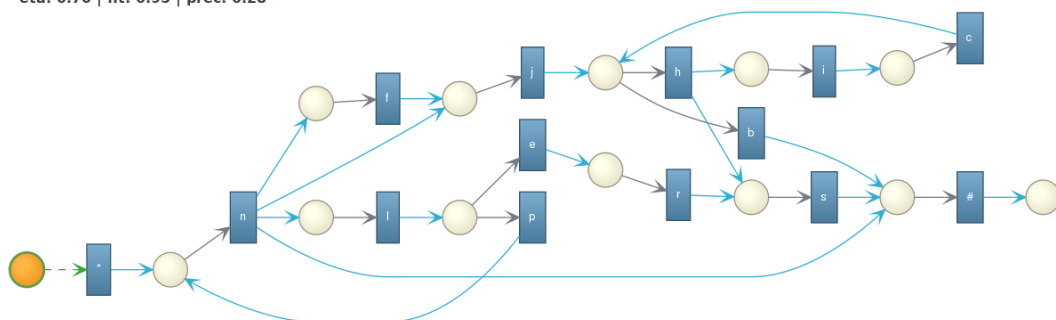
eta: 0.80 | fit: 0.96 | prec: 0.21

**2017.07**

eta: 0.70 | fit: 0.95 | prec: 0.32

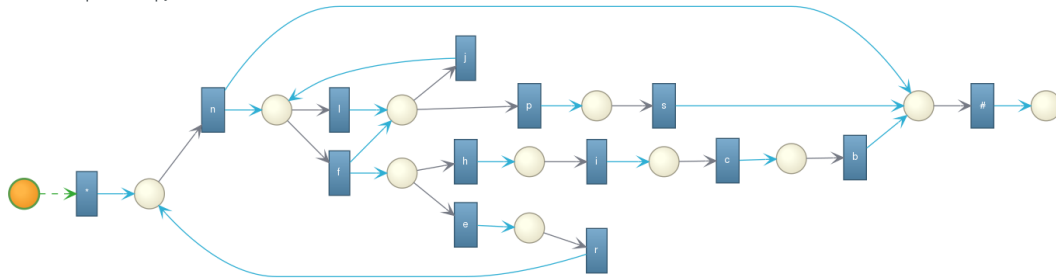
**2017.07**

eta: 0.70 | fit: 0.95 | prec: 0.28

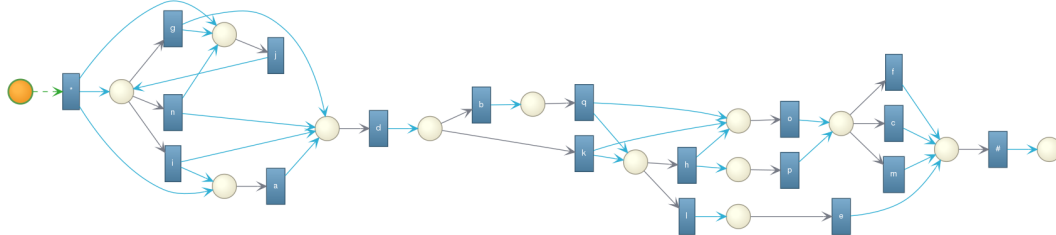


**2017.07**

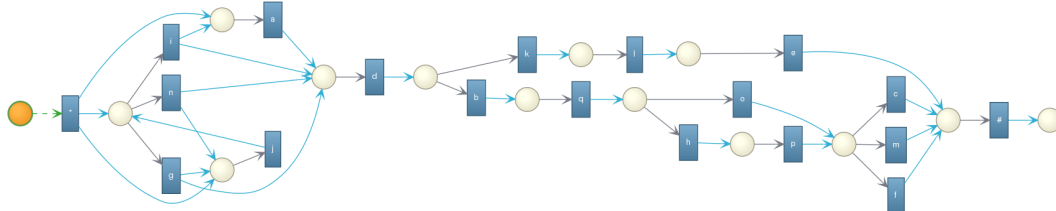
eta: 0.60 | fit: 0.94 | prec: 0.32

**2017.08**

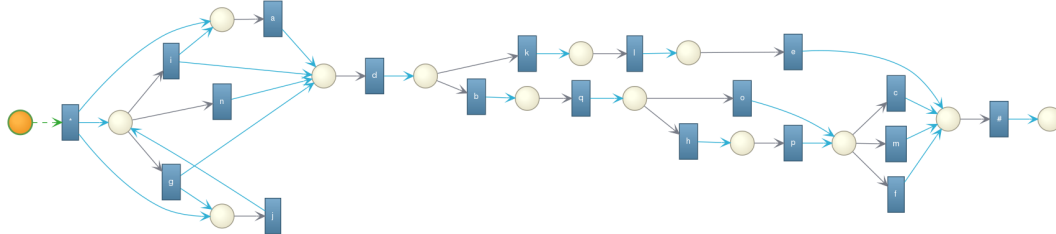
eta: 1.00 | fit: 0.91 | prec: 0.15

**2017.08**

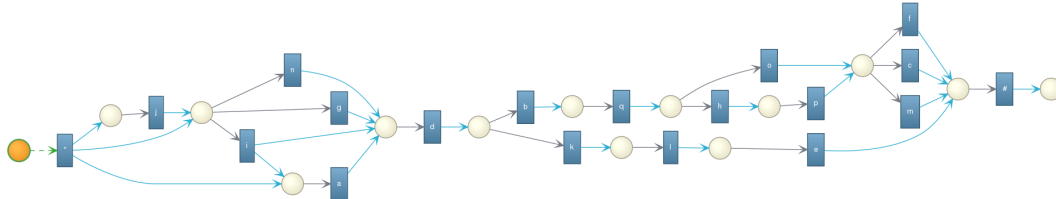
eta: 1.00 | fit: 0.89 | prec: 0.23

**2017.08**

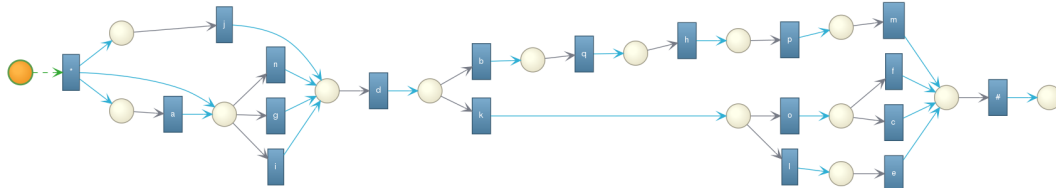
eta: 1.00 | fit: 0.89 | prec: 0.19

**2017.08**

eta: 1.00 | fit: 0.89 | prec: 0.20

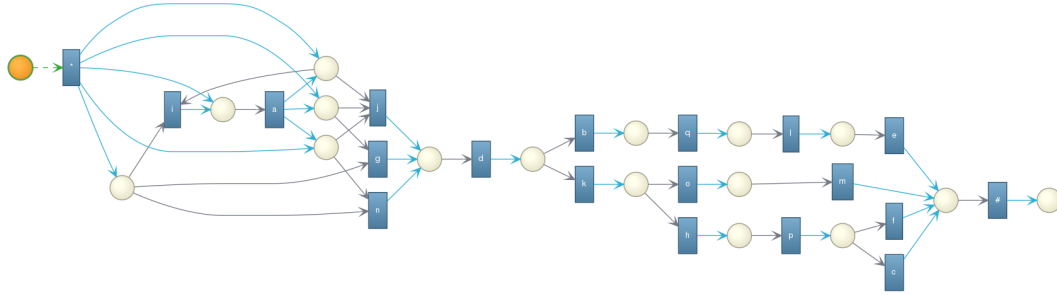
**2017.08**

eta: 1.00 | fit: 0.90 | prec: 0.23

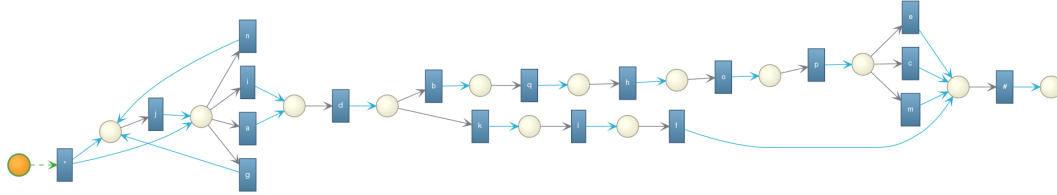


**2017.08**

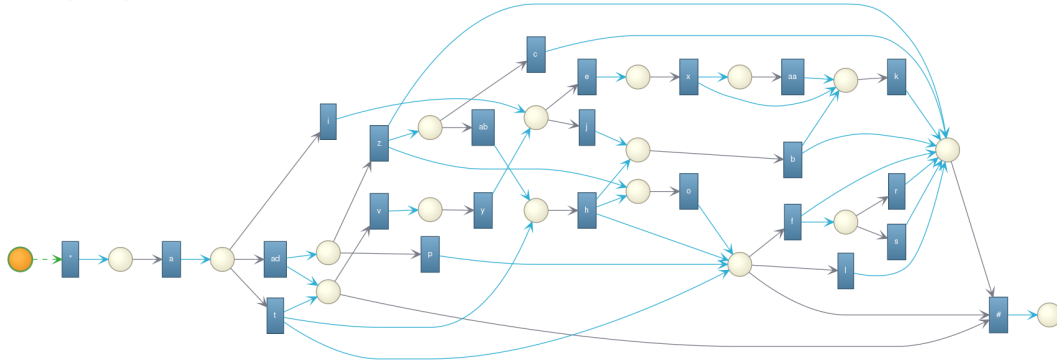
eta: 0.90 | fit: 0.89 | prec: 0.41

**2017.08**

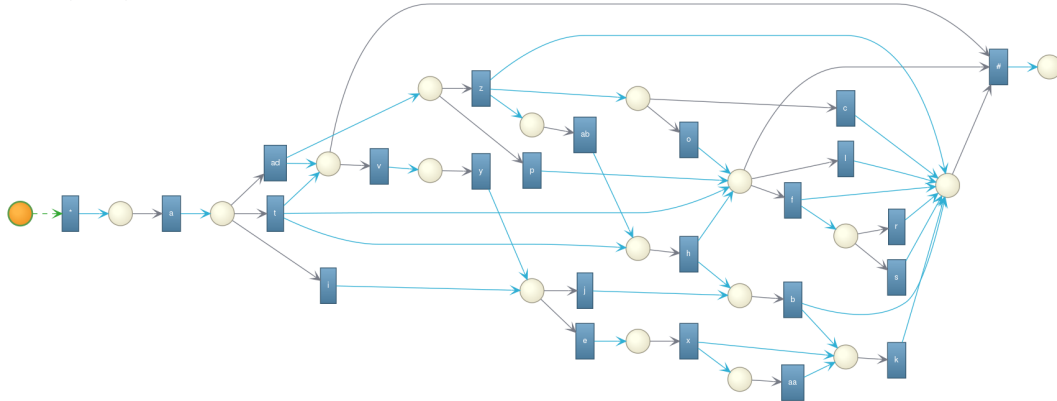
eta: 0.80 | fit: 0.90 | prec: 0.37

**2017.09**

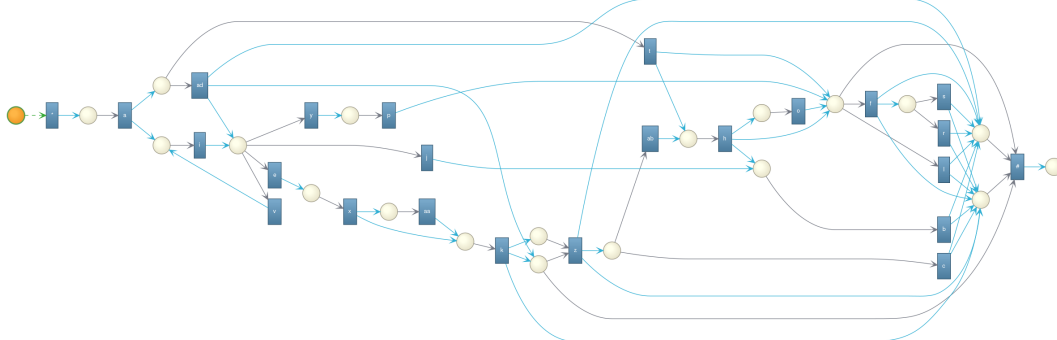
eta: 1.00 | fit: 0.79 | prec: 0.33

**2017.09**

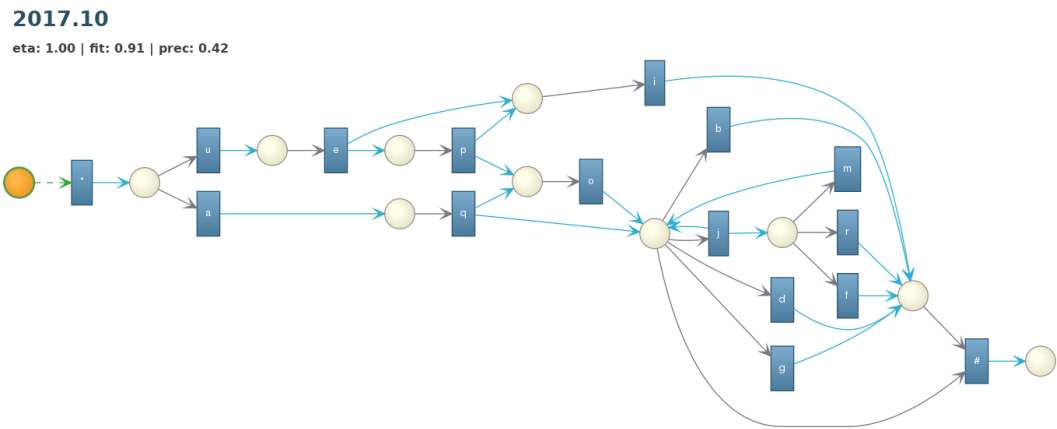
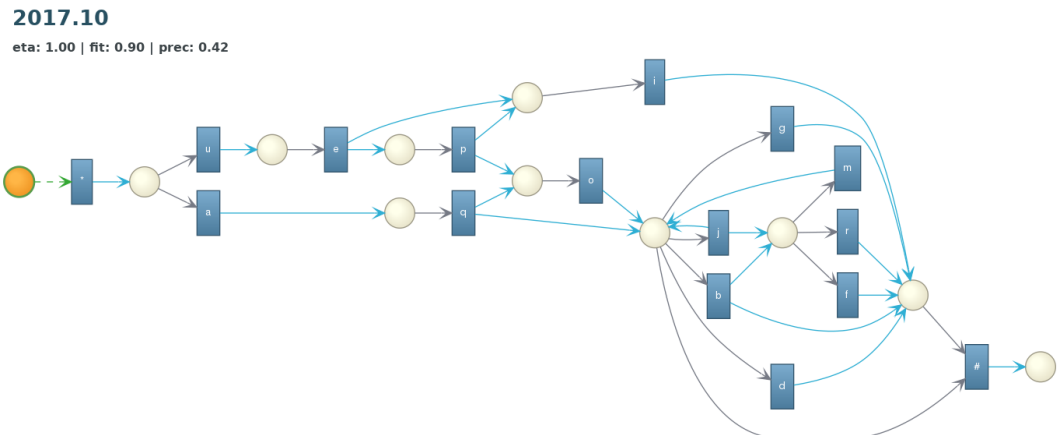
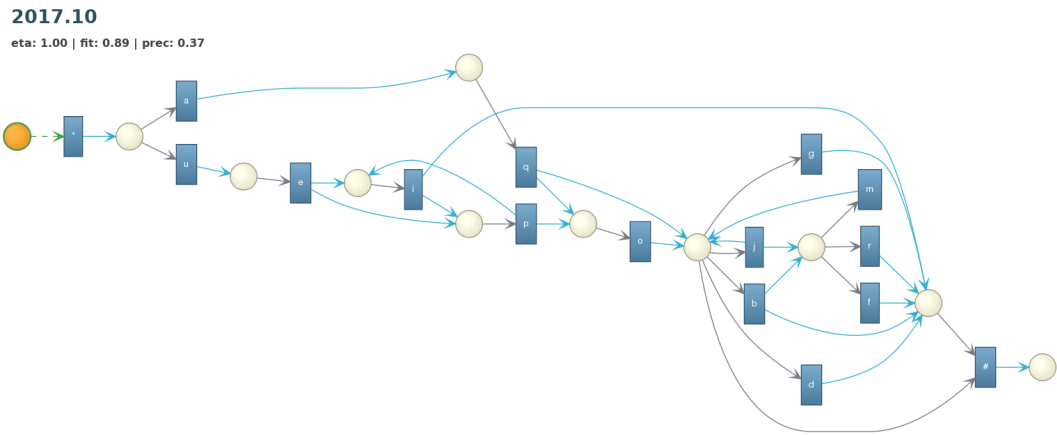
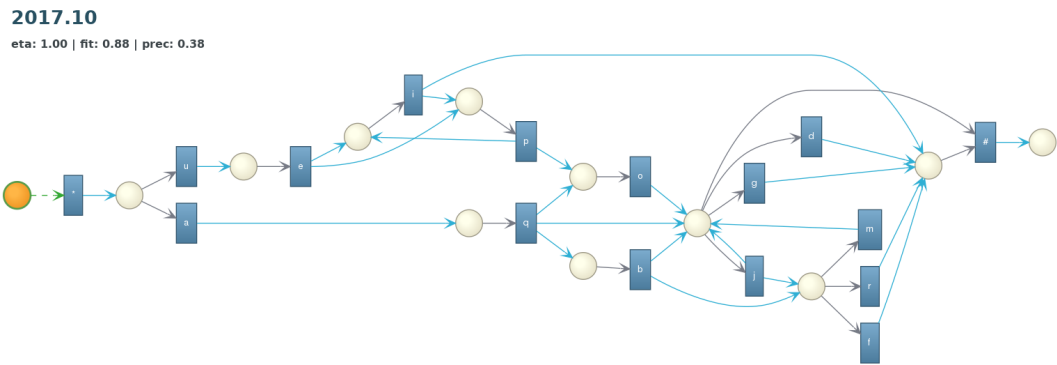
eta: 1.00 | fit: 0.79 | prec: 0.33

**2017.09**

eta: 0.80 | fit: 0.87 | prec: 0.22

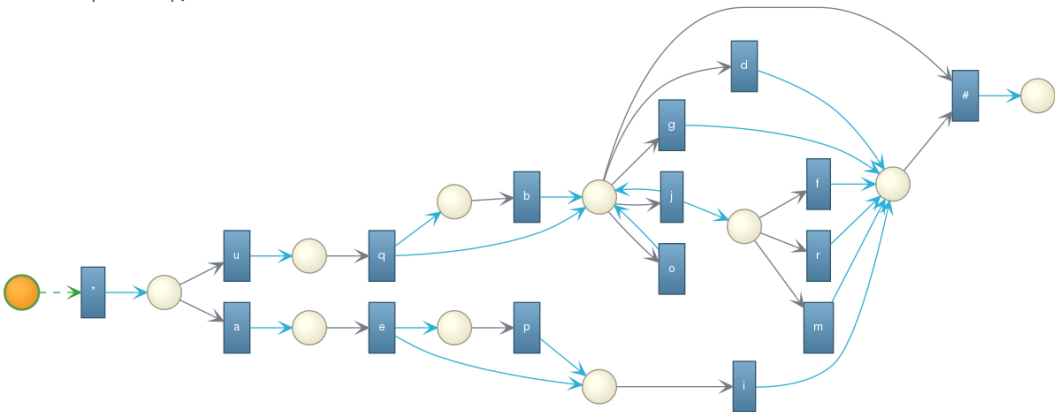






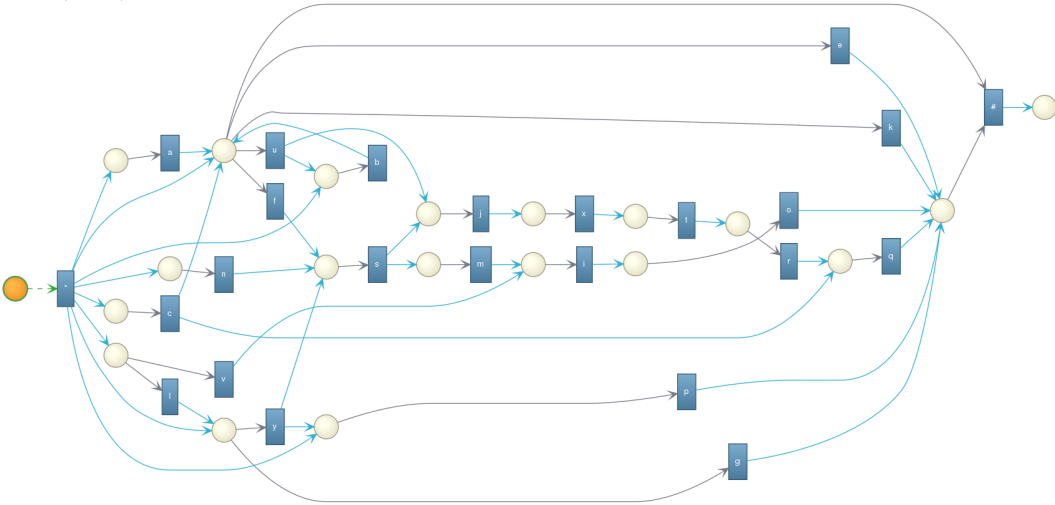
2017.10

eta: 0.90 | fit: 0.87 | prec: 0.56



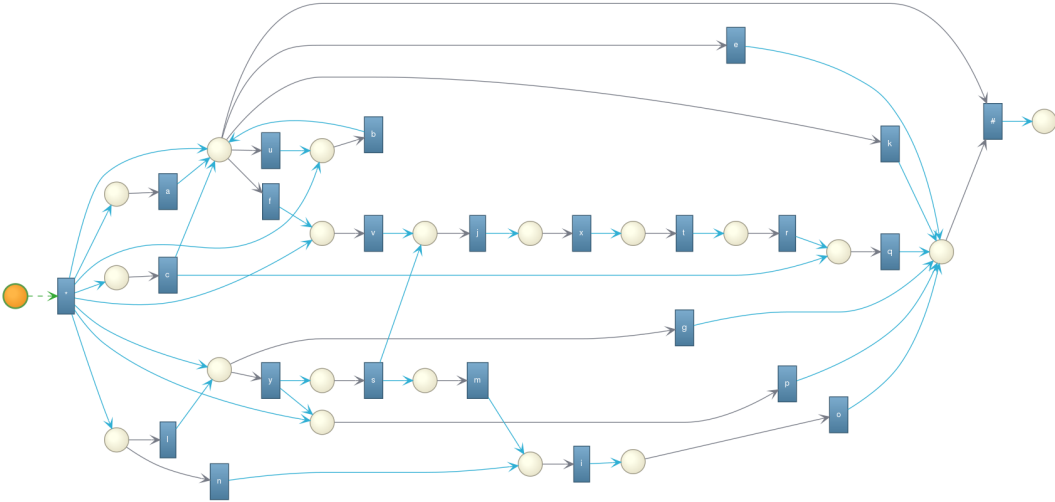
2016.03

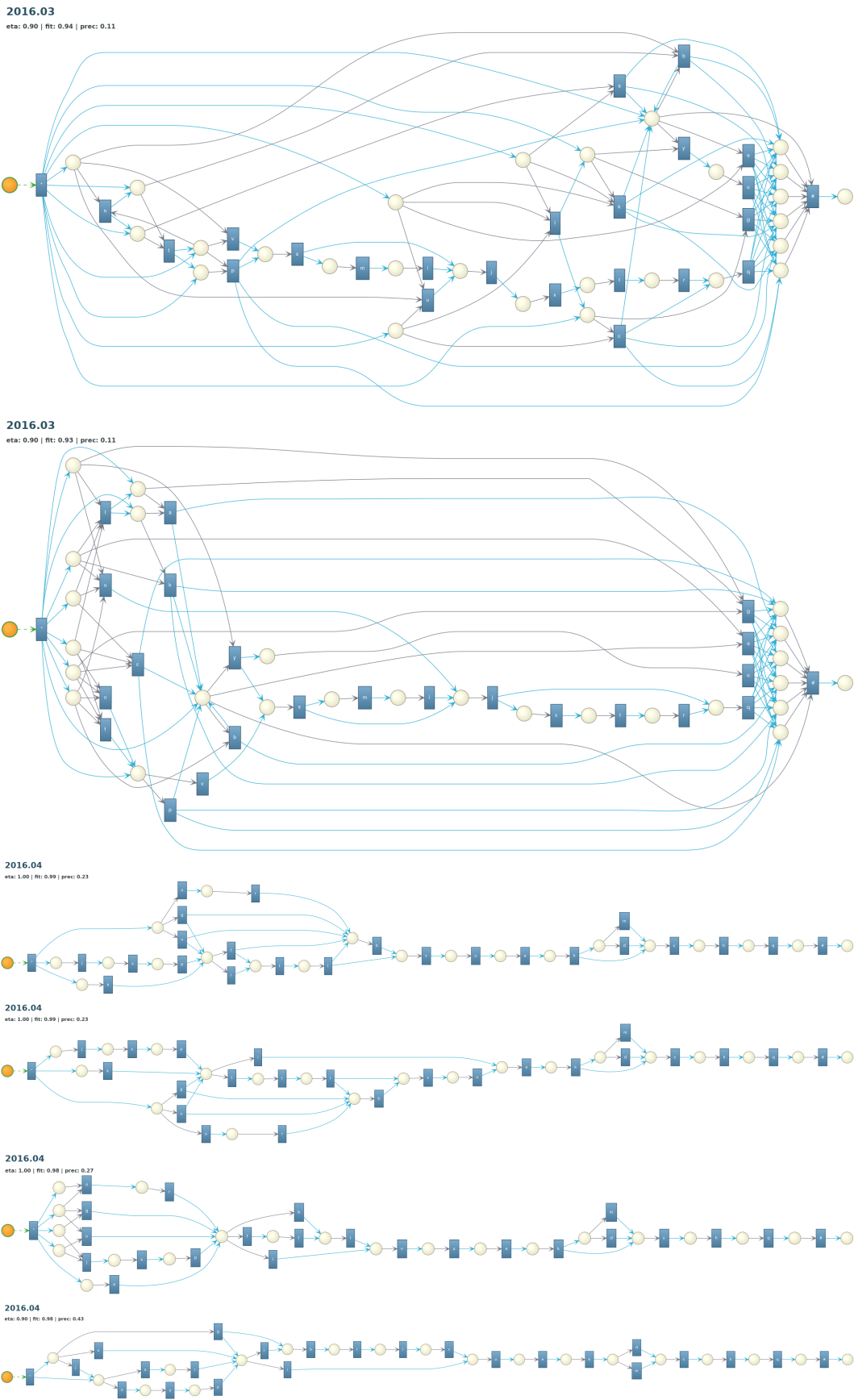
eta: 1.00 | fit: 0.93 | prec: 0.08

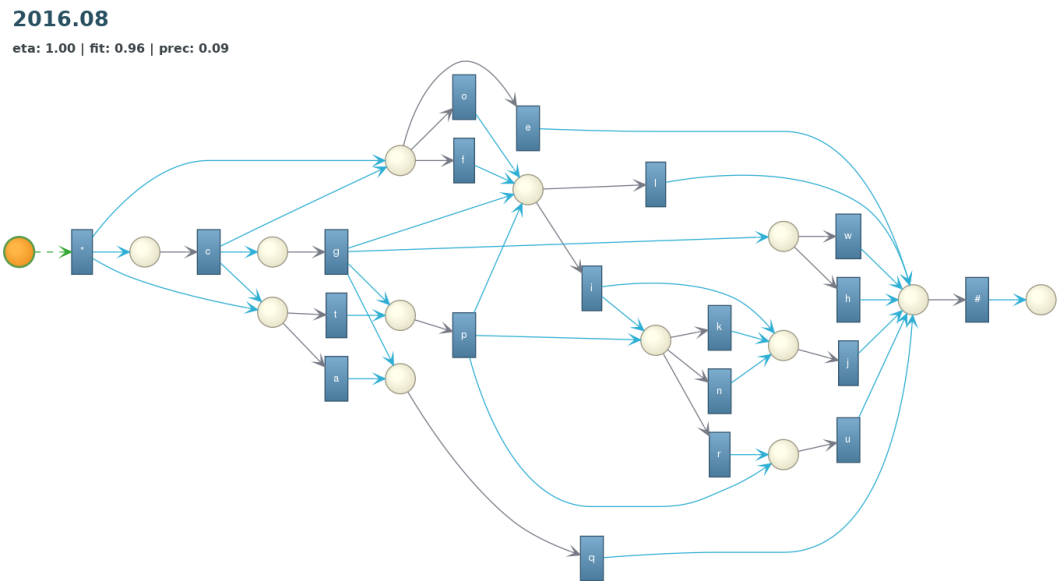
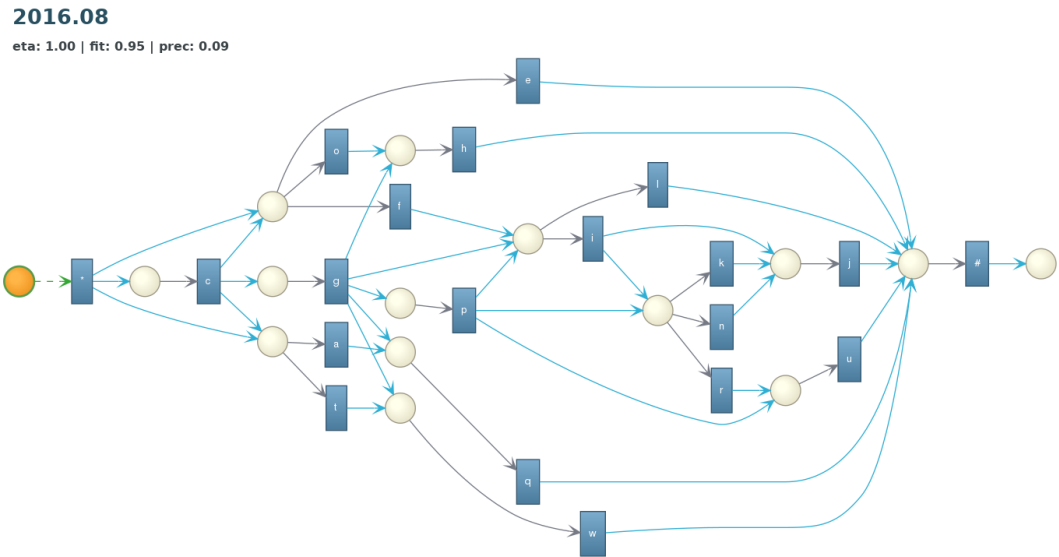
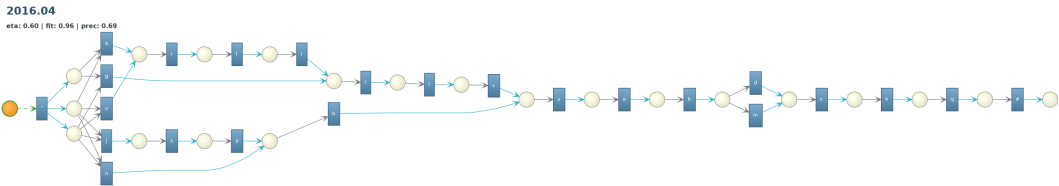
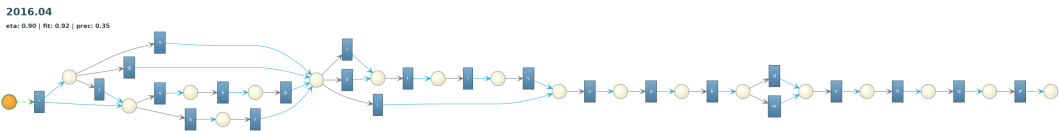


2016.03

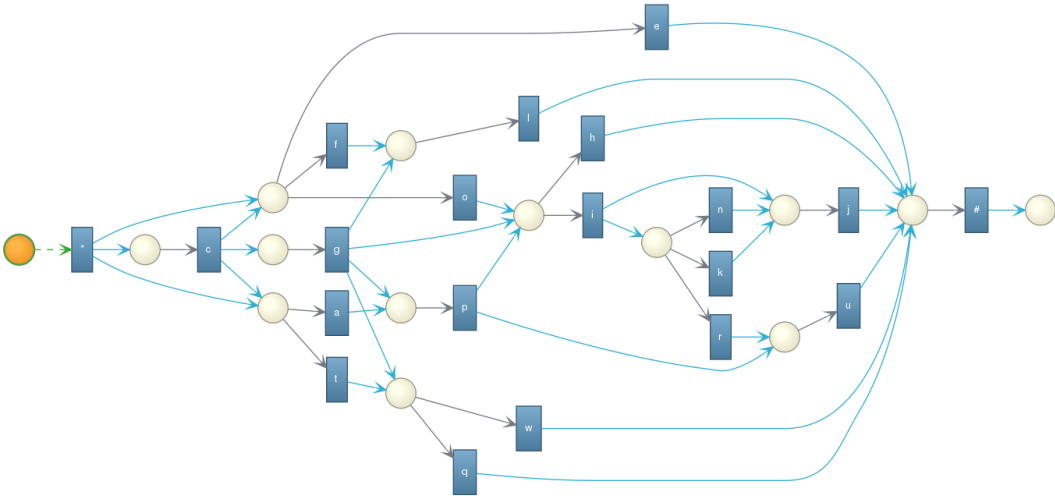
eta: 1.00 | fit: 0.93 | prec: 0.08



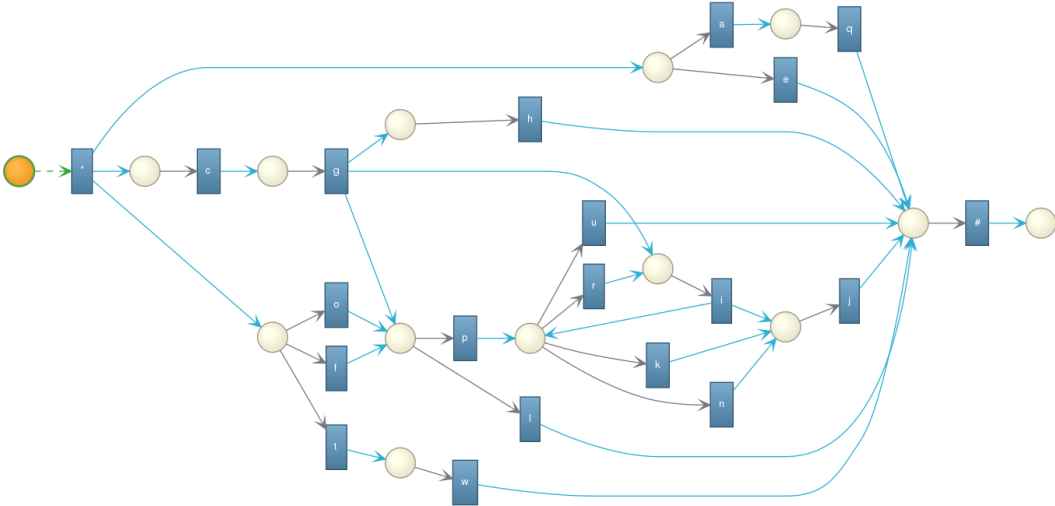




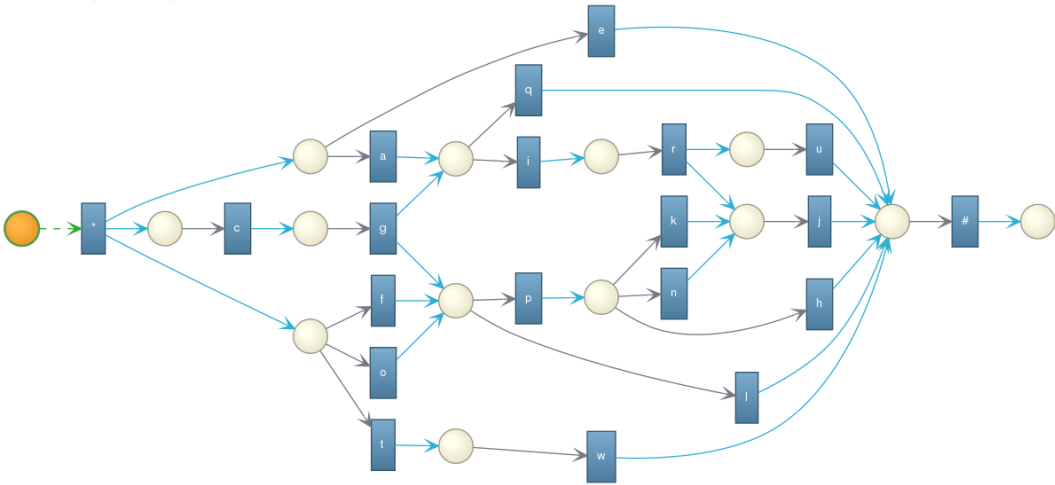
2016.08  
eta: 1.00 | fit: 0.96 | prec: 0.09



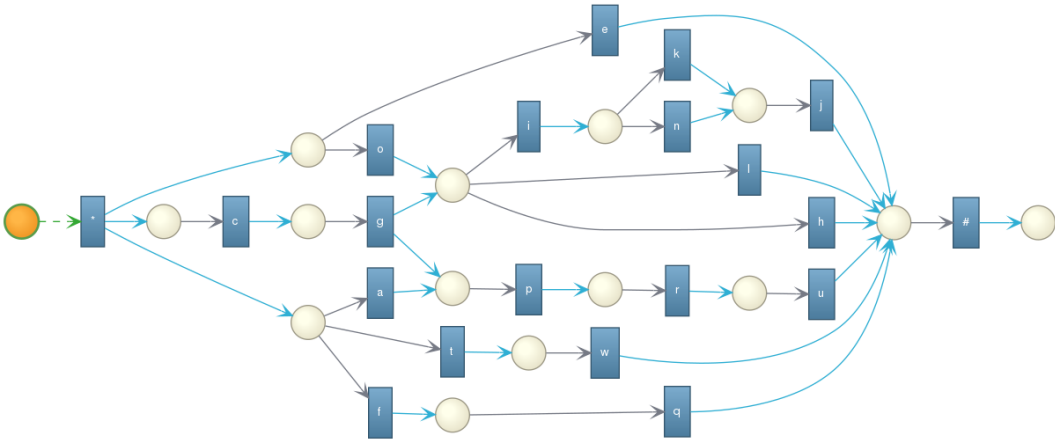
2016.08  
eta: 0.90 | fit: 0.96 | prec: 0.15



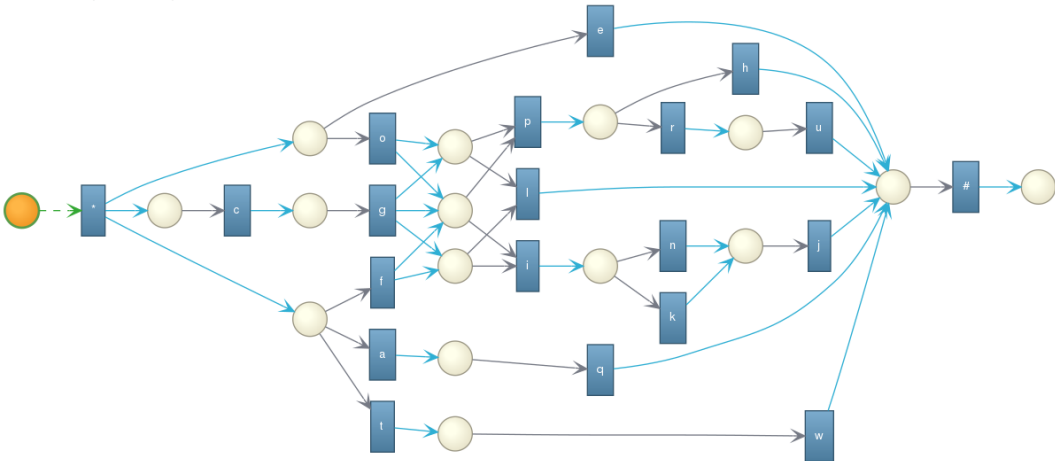
2016.08  
eta: 0.80 | fit: 0.95 | prec: 0.17



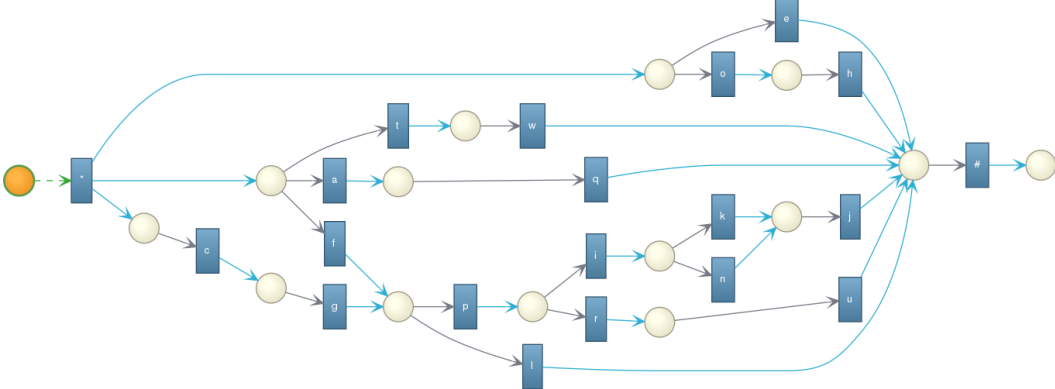
2016.08  
eta: 0.70 | fit: 0.94 | prec: 0.19



2016.08  
eta: 0.70 | fit: 0.95 | prec: 0.22




2016.08  
eta: 0.70 | fit: 0.93 | prec: 0.25



The diagram illustrates a deep neural network architecture for character recognition. It begins with an input layer (orange circle) that feeds into a convolutional layer (blue squares with letters). This layer is followed by a pooling layer (yellow circles). The network then branches into multiple parallel paths, each representing a different character (e.g., 'e', 'a', 't', 'f', 'o', 'c', 'l', 'p', 'q', 'i', 'r', 'u', 'w', 'h'). These paths converge into a final output layer (blue squares with letters), which is connected to a final output node (yellow circle). The network is designed to recognize characters from a set of 26 letters.

[illegible]

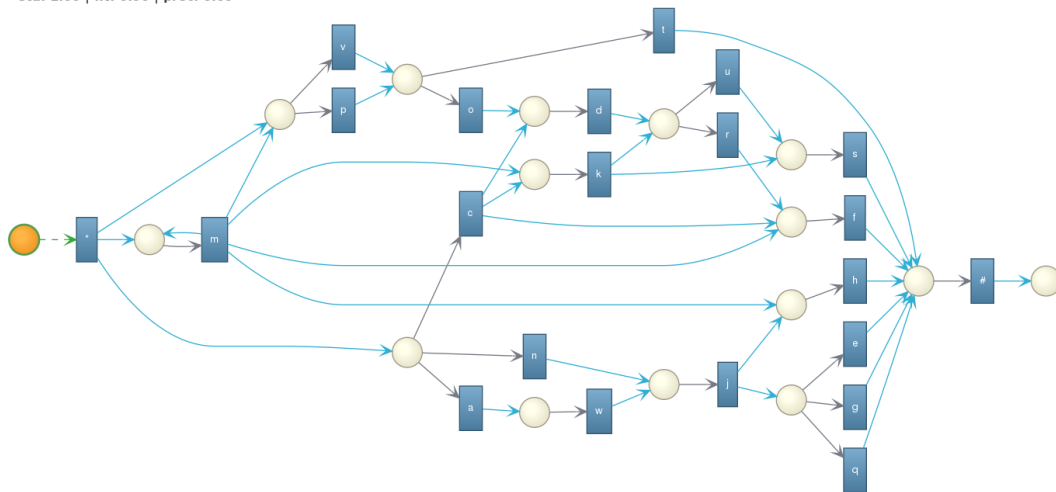
total: 0.70 | fit: 0.93 | prec: 0.70



total: 0.70 | fit: 0.93 | prec: 0.70

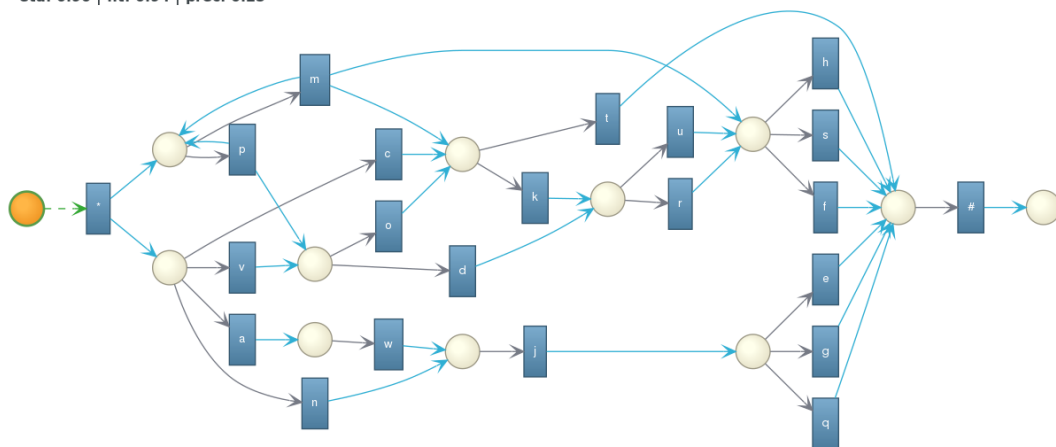
2016.10

eta: 1.00 | fit: 0.90 | prec: 0.09



2016.10

eta: 0.90 | fit: 0.94 | prec: 0.23



2016.10

eta: 0.60 | fit: 0.93 | prec: 0.14

