PUC

ISSN 0103-9741

Monografias em Ciência da Computação
n° 02/18

A global integer programming formulation for process discovery

Georges Miranda Spyrides

Beatriz Santiago

Marcus Poggi

Hélio Lopes

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22451-900

RIO DE JANEIRO - BRASIL

Monografias em Ciência da Computação, No. 02/18 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena May, 2018

A global integer programming formulation for process discovery

Georges Miranda Spyrides, Beatriz Santiago, Marcus Poggi, Hélio Lopes

gspyrides@inf.puc-rio.br, bsantiago@inf.puc-rio.br , poggi@inf.puc-rio.br, helio@inf.puc-rio.br

Abstract. Process Discovery amounts to determine a process model from an event log of a business process. Using the model obtained, one should be able to produce logs from the considered business process simulating the model. We propose an integer programming formulation that, given a log, determines all places and arcs defining a Petri net. Formulations from previous research discover one place at a time. To do so, we extend the ILP model in van der Werf et al. 2008 to consider global properties such as token balance and cohesion among places. Furthermore, the global approach allows more control of the Petri net properties: fitness, simplicity, generalization, and precision. We test the resulting methodology on event logs that address most of the pitfalls in process discovery algorithms. Also, we show the limitations of the method, regarding the Petri net morphology and log scale, and paths for its improvement.

Keywords: process discovery, integer linear programming.

In charge of publications:

Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC-Rio Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22451-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3527-1516 Fax: +55 21 3527-1530
E-mail: bib-di@inf.puc-rio.br
Web site: http://bib-di.inf.puc-rio.br/techreports/

Conteúdo

1 Introduction

Process discovery can be seen as a set of techniques comprised in the Process Mining community's accumulated knowledge. The main task executed by process discovery algorithms is to reconstruct a graphical representation of a business process from its event log.

One of the most common representations of business processes are Petri nets. A Petri net is a directed bipartite graph containing a set of transitions and a set of places that are connected between themselves. Petri nets have some simulation rules that provide interesting properties for users. For readers interested in deepening their understanding, we suggest the read of reference [4]. Therefore, in the context of this research, we are interested in deriving a Petri net from and event log. The Petri net transition set is immediately associate to the activities present in the event log, Methods to construct Petri nets to represent business processes concentrate on finding its places along with their connections.

Bergenthum et al [1] defines the relation between the theory of regions and formulation using the prefix-closed language associated 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. [7] and [8]. 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 for 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 don't deal with a wide variety of activities [9].

Often, researchers list two significant problems with the ILP approach: the complexity of the models and the scalability with a large variety of events. In [2], the authors compare several algorithms regarding model complexity. Among these, ILP miner consistently produces process models with more transitions and places than block-based methods such as the inductive miner. In [9], Zelst, Dongen, and Aalst, deal with the overfitting using sequence encoding filtering, thus removing infrequent behavior from the log.

Verbeek, in [12] and [11], addresses the scaling problem. In both articles, the authors explore the efficiency problem by subdividing logs into chunks. The time needed by the miner to find solutions is greatly reduced. Another attempt at simplifying the model is made in [10]. It tries to solve differently, proposing a mix 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.

Our motivation is to design a global formulation that returns the whole process in one execution of the optimizer. Along the paper, we propose a basic formulation and some ideas to improve the result, these are pointers for further research. We do not expect a time-efficient algorithm. 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. Of course, this limitation impacts our results. However, we believe that with proper decomposition strategies and heuristics, the efficiency and range of application of this approach can be improved in the future.

2 Definitions

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 σ, 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 in be represented by A^*.

Bag 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 σ that belongs to X and all sequences σ_1 that are a prefix of σ, i.e. σ_1 is such that $\sigma_1.\sigma_2 = \sigma$, we write:

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

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 $\{(T \times P) \bigcup (P \times T)\}$. We denote this bipartite graph as $G = (P, T, F)$.

The simulation of a Petri net, or its execution, 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, this configures a state of the Petri net. A marked Petri net is a Petri net at a given state and is completely described by P, T, F and m. 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 is triggered 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 σ over T, i.e. $\sigma \in T^*$ is a firing sequence of this Petri net if there exists a sequence of markings $m_1, m_2, \ldots, m_{|\sigma|}$

such that for $\sigma = \langle t_1, \ldots, t_{|\sigma|} \rangle$, transition t_j is enabled by marking m_{j-1} resulting in marking m_j, for $j = 1, \ldots, |\sigma|$.

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.

3 Formulation

The Process Discovery problem seeks for 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 minimum number of links, connection between two activities, or minimum number of places. We formalize the proposed ILP model for the Process Discovery problem.

Input An alphabet A, a set of strings S on elements of A. A contains two special symbols * and # that correspond to the start and the end of all strings in S, respectively. * and # only appear once in any string s in S.

Output A Petri net with a set of transitions A such that it produces exactly S, minimizing some structural property of the Petri net (number of places, number of "links", etc.)

Next, the proposed ILP formulation is presented. As stated above, the ILP we devise has as solution space a set of Petri nets that potentially generates the activity/event log given. 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.

We first recall the ILP model in van de Werf et al. [8]. It follows the use of Language Based Theory of Regions in Bergethum et al. [1] and in Lorenz and Juhás [3]. While [1] proposes algorithms that obtains several places through the resolution of an integer program and strives to eliminate redundant places, [8] makes use of objective functions and additional constraints that controls 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 a desired Petri net. We proceed in this section presenting the formulation for a single place, as in [8], [1].

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

$$\begin{array}{lcl|c} x_* - y_a & \geq & 0 & < a > \\ x_* - y_a + x_a - y_b & \geq & 0 & < a, b > \\ x_* - y_a + x_a - y_b + x_b - y_c & \geq & 0 & < a, b, c > \end{array}$$

Says that, for each trace, for each string s, prefix of this trace, 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.

3.2 Proposed formulation's core ideas

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 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 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 $k = 1, \ldots, K$, as described above, and variables $z_{i,j}^k$, with $k = 1, \ldots, K$, and $w_{i,j}$, for all pairs (i, j) of activities. While variables $z_{i,j}^k$ 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:

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

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:

$$z_{ij}^k \leq x_i^k \quad \forall (i, j) \in A \times A, \ k = 1, \ldots, K \tag{2}$$

$$z_{ij}^k \leq y_j^k \quad \forall (i, j) \in A \times A, \ k = 1, \ldots, K \tag{3}$$

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

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 z with variables w.

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

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

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

$$w_{ij} \geq z_{ij}^k \qquad \forall (i,j) \in A \times A \wedge \forall k \in K \tag{6}$$

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 [6].

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:

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

3.3 Global Integer Linear Programming Formulation

The basic complete ILP model for process discovery can be presented. It consists of an objective function subject to constraints (1) - (7). There are two straightforward objective functions that lead to minimal Petri nets: minimize the number of links and minimize the number of places or arcs used by the places. Our tests balances those elements assigning weights to the use of the variables. Therefore:

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

3.4 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:

$$\begin{array}{rcl|c} x_c^k - y_\#^k & \geq & 0 & < c > \\ x_b^k - y_c^k + x_c^k - y_\#^k & \geq & 0 & < b, c > \\ x_a^k - y_b^k + x_b^k - y_c^k + x_c^k - y_\#^k & \geq & 0 & < a, b, c > \end{array} \tag{8}$$

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, assuming the beginning and the ending of a trace may reflect more accurately the real process than parts of trace string in its middle, the use of both of these constraint types are expected to allow the methodology to find Petri nets bound to be a better representation of the original process.

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.:

$$f : (A \times A) \to \mathbb{N} \tag{9}$$

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) [6]. 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 hyper parameters 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:

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

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 the the first activity of some trace in the event log. Also, $w_{j\#}$ is set to one if j when 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:

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

Token Symmetry: Another constraint has use to conform the resulting Petri net to the soundness property. 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:

$$\sum_{k=1}^{K} \sum_{i \in A} x_i^k = \sum_{k=1}^{K} \sum_{i \in A} y_i^k \tag{10}$$

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

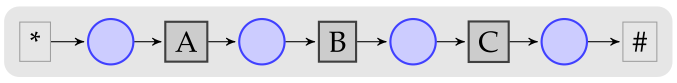
4 Model Evaluation and Experiment

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 a number of 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.

4.1 Artificial logs

We devised 14 small logs to reproduce common situations faced by process discovery algorithms. Their heterogeneity allowed us to observe 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.

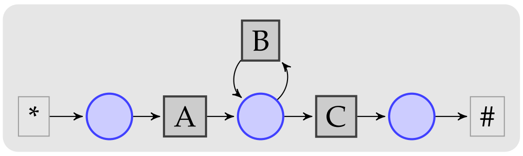


Log:

- ⟨A, B, C⟩

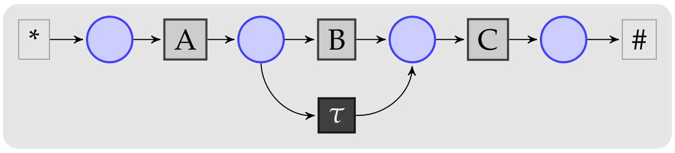
Log 2 - Skip/loop: Process with skip or optional simple loop. This example introduces one of the first difficulties on modeling. Once we work with few different traces, we may model this process as a single loop or as a skip.

Our formulation does not handle τ 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.

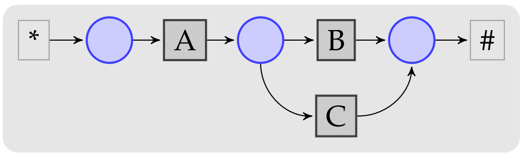


Log:

- ⟨A, B, C⟩,
- ⟨A, C⟩



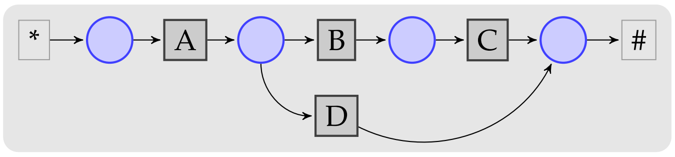
Log 3 - OR-open: Process with decision without closing. This log tests the ability to capture OR-gates without closing the flow.



Log:

- ⟨A, B⟩,
- ⟨A, C⟩

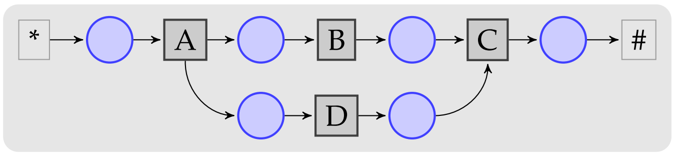
Log 4 - asym. OR-open Process with decision without closing 2. This log tests the capability of capturing OR-gates without closing the flow with a slight asymmetry.



Log:

- ⟨A, B, C⟩,
- ⟨A, D⟩

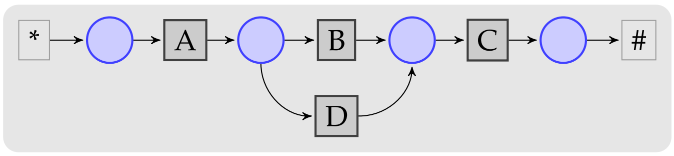
Log 5 - AND-open/close: Process with parallelism. In this log, we test the algorithm's capability of identifying all the multiple places that represent parallel flows. Traces in the log vary between themselves by changing the order of activities. Therefore, the algorithm must be capable of identifying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



Log:

- ⟨A, B, D, C⟩,
- ⟨A, D, B, C⟩

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.



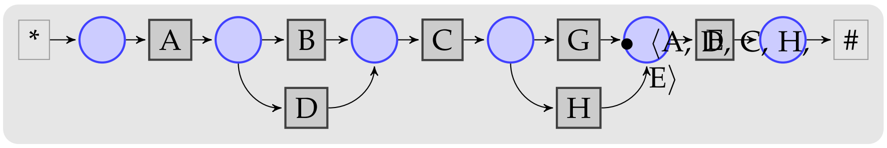
Log:

- ⟨A, B, C⟩,
- ⟨A, D, C⟩

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:

- ⟨A, B, C, G, E⟩,
- ⟨A, B, C, H, E⟩,
- ⟨A, D, C, G, E⟩,
- ⟨A, D, C, H, E⟩

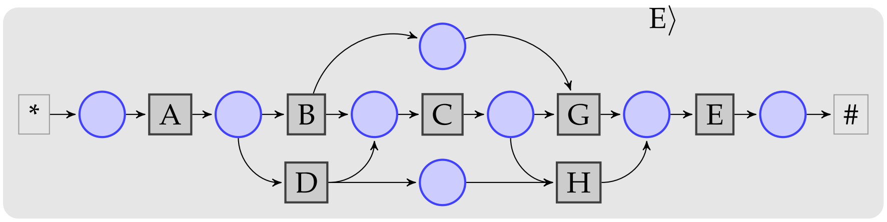


Log 8 - Choice-relation: Long-term choice relationship. This log is similar to the last one, but a more correct models bounds the decisions made in different points in the flow of the process.

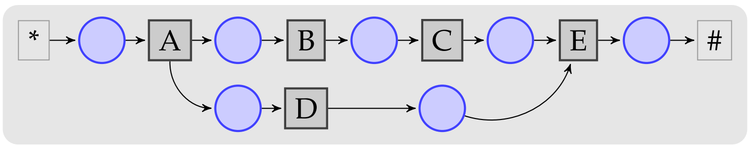
This means, there are two new places added to the last model to force a trace passing through B to pass through G later and, similarly, another place bounds D to H.

Log:

- ⟨A, B, C, G, E⟩,
- ⟨A, D, C, H, E⟩



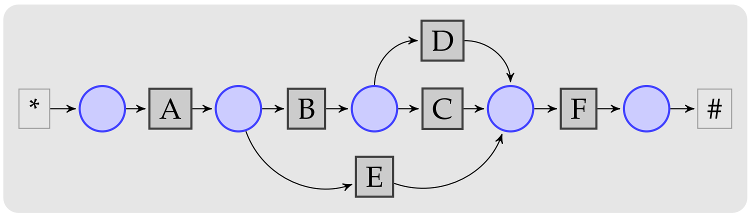
Log 9 - Asym. AND: Asymmetric parallelism. Just like the previous log, this one tests the ability of perceiving parallelism with slight scaling. This example introduces the difficulty of perceiving the consecutive relationship between activities B and C.



Log:

- ⟨A, B, C, D, E⟩,
- ⟨A, B, D, C, E⟩,
- ⟨A, D, B, C, E⟩

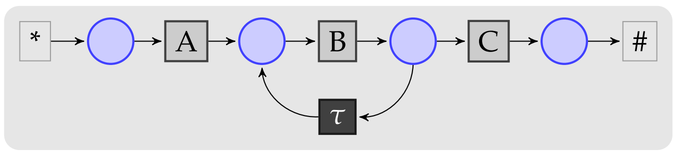
Log 10 - Nested 2OR: Nested decisions. This log increases the difficulty of previous decisions. It also has an interesting place with three entering arcs.



Log:

- ⟨A, B, C, F⟩,
- ⟨A, B, D, F⟩,
- ⟨A, E, F⟩

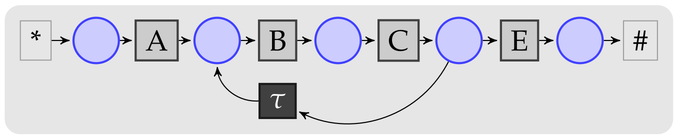
Log 11 - Oblig. Loop: Loop with at least one passage. This log represents a process with loop that must be executed at least one. It is different from the variant previously presented, where there was an option not execute the activity B.



Log:

- ⟨A, B, C⟩,
- ⟨A, B, B, C⟩,
- ⟨A, B, B, B, C⟩

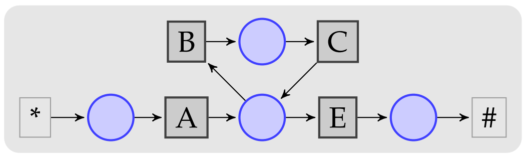
Log 12 - Oblig. Loop large; Larger loop with at least one passage. This log scales the concept introduced in the last one.



Log:

- ⟨A, B, C, E⟩,
- ⟨A, B, C, B, C, E⟩

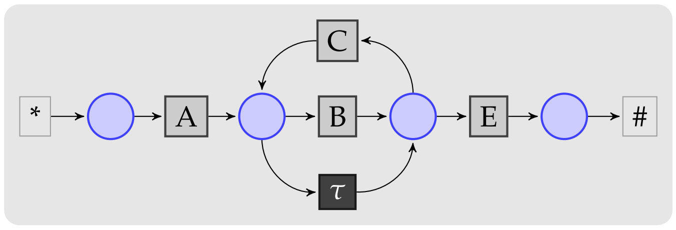
Log 13 - Optional loop: Larger optional loop. This log allows flow to ignore the loop.



Log:

- ⟨A, E⟩,
- ⟨A, B, C, E⟩,
- ⟨A, B, C, B, C, E⟩

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:

- ⟨A, E⟩,
- ⟨A, B, E⟩,
- ⟨A, B, C, B, E⟩

4.2 Experiment setups

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

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;

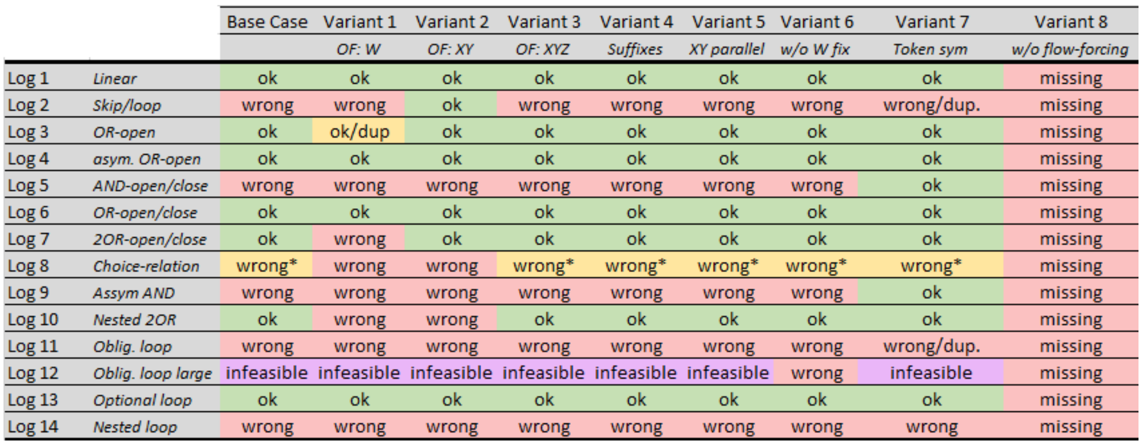
Variant 6 - w/o W fix. exclude Start and Finish fixing on the w_{ij} variables;

Variant 7 - token sym. forcing token symmetry;

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

5 Results

5.1 Correctness



		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

Abbreviations used in the table above:

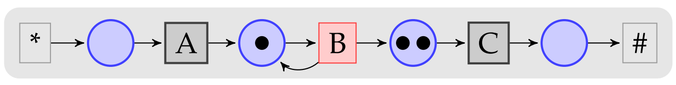
- **ok** - places found by the algorithm were expected;
- **wrong** - the algorithm found at least one wrong place;
- **dup** - some places found were duplicate;
- **missing** - the algorithm found some correct places but not all;
- **infeasible** - integer program generated does not have a feasible 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, the model finds only the start and finish places, which was fixed. Therefore, the correctness of all models rely on this set of constraints with exponential cardinality in relation to the input (number of cuts in the graph).

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 τ transition were all wrong. The formulation still does not provide a way to model τ. 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 work flow 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:

- ⟨A, B, C⟩,
- ⟨A, B, B, C⟩,
- ⟨A, B, B, B, C⟩

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 complex place that handles the sequence and the loop.

Some cycles of greater 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.

5.2 Time-efficiency

		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

The table above shows the execution time in seconds of the optimizers. In terms computing times, the only model that did poorly was the one without fixing start and end activities. The Logs that had a significant worse time-efficiency were the Logs 7 and 8. Interestingly, those have a greater variety of activities, which translate to more variables in the optimization model. [5]

6 Discussion

6.1 Findings and contributions

As expected the model does not handle situations where ghost (τ) 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.

The algorithm does not 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 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.

Prefix constraints and Suffix constraints may be redundant, as observed in the results from Variant 4 in relation to the Base Case. In this case, perhaps a good strategy would be to generate constraints only to prefixes that are about a bit more lengthy than half to size of the trace, and do the same with the suffixes. This would induce a formulation with way less non-zeros.

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.

6.2 Future research

We presented a powerful new idea for process discovery, although new developments are needed to scale to large models. Nevertheless, this idea can set a path for research on understanding the relation between logs and Petri nets' topology.

On the efficiency issue of our process discovery methodology, one clear acceleration may be achieved by decomposition where a master problem takes care of the global properties of the Petri net, while subproblems generate places. The use of polyhedral combinatorics to improve the linear programming relaxation of the model is other line for improvement.

Moreover, we observed that fixing the start and finish reduce the optimization time. We believe that there are opportunities in discovering heuristics that allow the model to fix more variables, especially the set of w_{ij} variables. This would lead also to better optimization performance.

Finally, a more immediate recommendation would be to model τ use cases. τ is 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.

Referências

[1] Robin Bergenthum, Jörg Desel, Robert Lorenz, and Sebastian Mauser. Process mining based on regions of languages. In *International Conference on Business Process Management*, pages 375–383. Springer, 2007.

[2] Sander JJ Leemans, Dirk Fahland, and Wil MP van der Aalst. Discovering block-structured process models from event logs containing infrequent behaviour. In *International Conference on Business Process Management*, pages 66–78. Springer, 2013.

[3] Robert Christian Lorenz, Sebastian Mauser, and Gabriel Juhás. How to synthesize nets from languages - a survey. *2007 Winter Simulation Conference*, pages 637–647, 2007.

[4] Wolfgang Reisig. *Understanding petri nets: modeling techniques, analysis methods, case studies.* Springer, 2013.

[5] Wil MP Van der Aalst. Process discovery: An introduction. In *Process Mining*, pages 125–156. Springer, 2011.

[6] W.M.P. van der Aalst. *Process Mining: Data Science in Action*. Springer Berlin Heidelberg, 2016.

[7] Jan Martijn EM Van der Werf, Boudewijn F van Dongen, Cor AJ Hurkens, and Alexander Serebrenik. Process discovery using integer linear programming. In *International conference on applications and theory of petri nets*, pages 368–387. Springer, 2008.

[8] J. M. E. M. van derWerf, B. F. van Dongen, C. A. J. Hurkens, and A. Serebrenik. Process discovery using integer linear programming. *Fundam. Inf.*, 94(3-4):387–412, August 2009.

[9] Sebastiaan J van Zelst, Boudewijn F van Dongen, and Wil MP van der Aalst. Avoiding over-fitting in ilp-based process discovery. In *International Conference on Business Process Management*, pages 163–171. Springer, 2015.

[10] Sebastiaan J van Zelst, Boudewijn F van Dongen, and Wil MP van der Aalst. Ilp-based process discovery using hybrid regions. In *ATAED@ Petri Nets/ACSD*, pages 47–61, 2015.

[11] HMW Verbeek and Wil MP van der Aalst. Decomposed process mining: The ilp case. In *International Conference on Business Process Management*, pages 264–276. Springer, 2014.

[12] HMW Verbeek, WMP van der Aalst, and J Munoz-Gama. Divide and conquer: a tool framework for supporting decomposed discovery in process mining. *The Computer Journal*, 60(11):1649–1674, 2017.