Process Mining Based on Regions of Languages

Robin Bergenthum, Jörg Desel, Robert Lorenz, and Sebastian Mauser

Department of Applied Computer Science,
Catholic University of Eichstätt-Ingolstadt
{firstname.lastname}@ku-eichstaett.de

Abstract. In this paper we give an overview, how to apply region based methods for the synthesis of Petri nets from languages to process mining.

The research domain of process mining aims at constructing a process model from an event log, such that the process model can reproduce the log, and does not allow for much more behaviour than shown in the log. We here consider Petri nets to represent process models. Event logs can be interpreted as finite languages. Region based synthesis methods can be used to construct a Petri net from a language generating the minimal net behaviour including the given language. Therefore, it seems natural to apply such methods in the process mining domain. There are several different region based methods in literature yielding different Petri nets. We adapt these methods to the process mining domain and compare them concerning efficiency and usefulness of the resulting Petri net.

1 Introduction

Often, business information systems log all performed activities together with the respective cases the activities belong to in so called event logs. These event logs can be used to identify the actual workflows of the system. In particular, they can be used to generate a workflow definition which matches the actual flow of work. The generation of a workflow definition from event logs is known as *process mining*. Application of process mining and underlying algorithms gained increasing attention in the last years, see e.g. [18] and [17]. There are a number of process mining tools, mostly implemented in the ProM framework [13].

The formal problem of generating a system model from a description of its behaviour is often referred to as synthesis problem. Workflows are often defined in terms of Petri nets [16]. Synthesis of Petri nets is studied since the 1980s [8,9]. Algorithms for Petri net synthesis have often been applied in hardware design [5]. Obviously, process mining and Petri net synthesis are closely related problems. Mining aims at a system model which has at least the behaviour given by the log and does not allow for much more behaviour. In the optimal case the system has minimal additional behaviour. The goal is to find such a system which is not too complex, i.e., small in terms of its number of components. This is necessary, because practitioners in industry are interested in controllable and interpretable reference models. Apparently, sometimes a trade-off between the size of the model and the additional behaviour has to be found.

One of the main differences in Petri net synthesis is that one is interested in a Petri net representing exactly the specified behaviour. Petri net synthesis was originally assuming a behavioural description in terms of transition systems. For a transition system, sets of nodes called *regions* can be identified. Each region refers to a place of the synthesized net. Analogous approaches in the context of process mining are presented in [19,15]. Since process mining usually does not start with a transition system, i.e., a state based description of behaviour, but rather with a set of sequences, i.e., a language based description of behaviour, the original synthesis algorithms are not immediately applicable. In [19,15] artificial states are introduced into the log in order to generate a transition system. Then synthesis algorithms transforming the state-based model into a Petri net, that exactly mimics the behaviour of the transition system, are applied. The problem is that these algorithms include reproduction of the state structure of the transition system, although the artificial states of the transition system are not specified in the log. In many cases this leads to a bias of the process mining result. However, there also exist research results on algorithmic Petri net synthesis from languages [6,1,2,10]. In these approaches, regions are defined on languages. It seems natural to directly use these approaches for process mining, because logs can directly be interpreted as languages. The aim of this paper is to adjust such language based synthesis algorithms to solve the process mining problem. This approach is very well suited for process mining, because wether or not the synthesized net exactly represents the given language, it always reproduces the language (given by an event log).

We present and compare methods for process mining adapted from language based synthesis and give a complete overview of the applicability of regions of languages to the process mining problem. Finally, we provide a bridge from the more theoretical considerations of this paper to practically useful algorithms. The process mining algorithms discussed in this paper are completely based on formal methods of Petri net theory guaranteeing reliable results. By contrast, most existing process mining approaches are partly based on heuristic methods, although they borrow techniques from formally developed research areas such as machine learning and grammatical inference [18,12], neural networks and statistics [18,4], or Petri net algorithms [7,19,15].

We omitted formal definitions, lemmas, theorems and proofs in this short paper. These are provided by the technical report [3]. In [3] the interested reader can also find more detailed explanations and pseudo code of the developed algorithms.

2 Application of Regions of Languages to Process Mining

First we introduce the process mining problem and show how the classical language based theory of regions [6,1] can be adapted to solve this problem. Process mining aims at the construction of a process model from an *event log* which is able to reproduce the behaviour (the process) of the log, and does not allow for much more behaviour than shown in the log. The following example log σ will serve as a running example. Since we focus on the control flow of activities (their ordering), we abstract from some additional log information such as originators of events and time stamps of events. The control flow, i.e. the behaviour, of the event log is given by a prefix-closed finite language over the alphabet of activities, the so called *process language* $L(\sigma)$.

event log (activity,case):
(a,1) (b,1) (a,2) (b,1) (a,3) (d,3) (a,4) (c,2) (d,2) (e,1) (c,3) (b,4) (e,3) (e,2) (b,4) (e,4)
process language:
a ab abb *abbe* ac acd *acde* ad adc *adce*

Figure 1 shows a *marked place/transition-net (p/t-net)* (N, m_0) having exactly the process language $L(\sigma)$ as its *language of occurrence sequences* $L(N, m_0)$. That means this Petri net model is a process model describing the process given by the event log σ.

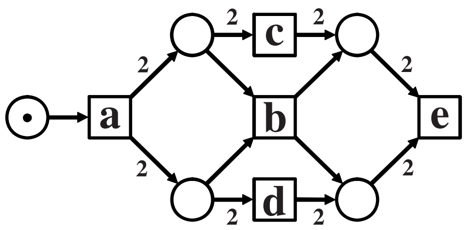

Fig. 1. Petri net model fulfilling $L(N, m_0) = L(\sigma)$

The process model in the ideal case serves as a reference model interpretable by practitioners. Therefore the model should be as small as possible. As we will show, there is a trade-off between the size of the constructed model and the degree of the match of the behaviour generated by the model and the log. In this paper we formalize process models as Petri nets and consider the following *process mining problem*:

Given: An event log σ. **Searched:** A preferably small finite marked p/t-net (N, m_0) such that **(1)** $L(\sigma) \subseteq L(N, m_0)$ and **(2)** $L(N, m_0) \setminus L(\sigma)$ is small.

In the following we will consider a fixed process language $L(\sigma)$ given by an event log σ with set of activities T. An adequate method to solve the process mining problem w.r.t. $L(\sigma)$ is applying synthesis algorithms using regions of languages: The set of transitions of the searched net (N, m_0) is given by the set of characters T used in $L(\sigma)$. The behaviour of this net is restricted by adding places. Every place is defined by its initial marking and the weights of the arcs connecting them to each transition $t \in T$. In order to guarantee (1), i.e. to reproduce the log, only places are added, which do not prohibit sequences of $L(\sigma)$. Such places are called *feasible (w.r.t.* $L(\sigma)$*)*. The more feasible places we add the smaller is the set $L(N, m_0) \setminus L(\sigma)$. Adding *all* feasible places minimizes $L(N, m_0) \setminus L(\sigma)$ (preserving (1)). That means the resulting net – called the *saturated feasible net* – is an optimal solution for the process mining problem concerning (1) and (2). But it is not small, even not finite. Here the trade-off between the size of the constructed net and (2) comes into play: The more feasible places we add the better (2) is reached, but the bigger becomes the constructed net. The central question is which feasible places should be added. Two procedures are candidates to solve this problem: There are two basic algorithmic approaches throughout the literature to synthesize a finite net (N, m_0) from a finite language. The crucial idea in these approaches is to define feasible places structurally on the level of the given language: Every feasible place is defined by a so called *region* of the language. A region is simply a $(2|T| + 1)$-tuple of natural numbers which represents the initial marking of a place and the number of tokens each transition consumes respectively produces in that place, satisfying some property which ensures that no occurrence sequence of the given (process) language $L(\sigma)$ is prohibited by this place. The set of regions can be characterized as the set of non-negative integral solutions of a homogenous linear inequation system $\mathbf{A}_{L(\sigma)} \cdot \mathbf{r} \geq \mathbf{0}$ (with integer coefficients) having $|L(\sigma)|$ rows. Both approaches use linear programming techniques and convex geometry to calculate a certain adequate finite set of solutions of this system. In the following we adjust both procedures to the considered process mining problem and discuss their applicability and their results in this context.

The first strategy to add a certain finite set of feasible places, used in [10], computes a so called *finite basis* of the set of all feasible places (any feasible place is a non-negative linear combination of the basis). Adding all basis places leads to a finite representation of the saturated feasible net. Consequently, this approach leads to an optimal solution of the process mining problem concerning (2). The set of regions is given by the integer points of a pointed *polyhedral cone* [14]. The finite set of rays of the cone leads to a (minimal) basis of the set of regions and thus defines a finite basis of the set of feasible places [3,14]. It can be effectively computed from $\mathbf{A}_{L(\sigma)}$ (see for example [11]). The time complexity of the computation essentially depends on the number k of basis regions which is bounded by $k \leqslant \binom{|L(\sigma)|+2|T|+1}{2|T|+1}$. That means, in the worst case the time complexity is exponential in $|L(\sigma)|$, whereas in most practical examples the number of basis solutions is reasonable. The calculated finite set of basis places usually still includes so called redundant places, which can be omitted from the net without changing its language of occurrence sequences. Some of these redundant places can easily be identified [3]. These are finally deleted from the constructed net. The resulting process mining algorithm, called method 1 in the following, is shown in [3].

For the event log of the running example, method 1 computes 55 basis places (corresponding to rays). 15 of these places are directly deleted as easily identifiable redundant places. Many of the 40 places of the resulting net are still redundant. It is possible to calculate a minimal subset of places generating the same language of occurrence sequences. This would lead to the net shown in Figure 1 with only five key places. But this is extremely inefficient. Thus, more efficient heuristic approaches to delete redundant places are of interest. The practical applicability of the algorithm could be drastically improved with such heuristics. In the considered example, most of the redundant places are so called loop places. If we delete all loop places from the constructed net with 40 places, there remain the five places shown in Figure 1 plus the eight redundant places shown in Figure 2. In this case this procedure did not change the behaviour of the net.

In this example the process language is exactly reproduced by the constructed net. Usually this is not the case. For example omitting the word $acde$ (but not its prefixes) from the process language, the inequation system is not changed. Therefore the net constructed from this changed language with method 1 coincides with the above example. This net has the additional occurrence sequence $adce$ not belonging to the changed process language. Since the net calculated by method 1 is the best approximation to the given language, the changed process language (given by a log) has to be completed in this way to be describable as a p/t-net.

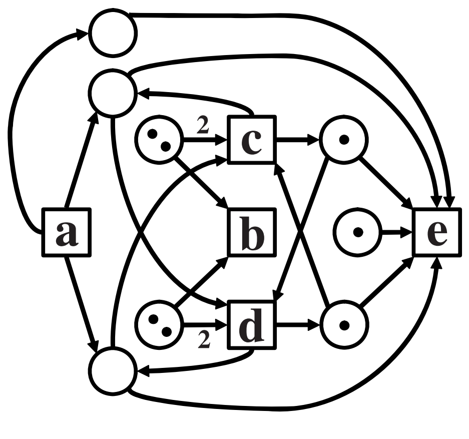

Fig. 2. Redundant places computed with method 1

The main advantage of method 1 is the optimality w.r.t. (2). The resulting process model may be seen as a natural completion of the given probably incomplete log file. Problematic is that the algorithm in some cases may be inefficient in time and space consumption. Moreover, the resulting net may be relatively big.

The second strategy to synthesize a finite net, used e.g. in [1,2], is to add such feasible places to the constructed net, which *separate* specified behaviour from non-specified behaviour. That means for each $w \in L(\sigma)$ and each $t \in T$ such that $wt \notin L(\sigma)$, one searches for a feasible place p_{wt}, which prohibits wt. Such wt is called *wrong continuation* (also called faulty word in [1]) and such places are called *separating feasible places*. If there is such a separating feasible place, it is added to the net. The number of wrong continuations is bounded by $|L(\sigma)| \cdot |T|$. Thus the set containing one separating feasible place for each wrong continuation, for which such place exists, is finite. The net resulting from adding such a set of places yields a good solution for the process mining problem: If the process language of the log can exactly be generated by a p/t-net, the constructed net is such a net. Consequently, in this case (2) is optimized. In general (2) is not necessarily optimized, since it is possible that even if there is no feasible place prohibiting wt, there might be one prohibiting wtt' – but such places are not added. However, in most practical cases this does not happen [3].

In order to compute a separating feasible place which prohibits a wrong continuation wt, one defines so called *separating regions* defining such places. These are defined by one additional (strict) inequation ensuring that wt is prohibited. Thus a separating region $\mathbf{r}$ w.r.t. a wrong continuation wt can be calculated (if it exists) as a non-negative integer solution of a homogenous linear inequation system with integer coefficients of the form $\mathbf{A}_{L(\sigma)} \cdot \mathbf{r} \geq \mathbf{0}, \mathbf{b}_{wt} \cdot \mathbf{r} < \mathbf{0}$. The matrix $\mathbf{A}_{L(\sigma)}$ is defined as before. If there exists no non-negative integer solution of this system, there exists no separating region w.r.t. wt and thus no separating feasible place prohibiting wt. If there exists a non-negative integer solution of the system, any such a solution defines a separating feasible place prohibiting wt.

There are several linear programming solver to decide the solvability of such a system and to calculate a solution if it is solvable. The choice of a concrete solver is a parameter of the process mining algorithm, that can be used to improve the results or the runtime. Since the considered system is homogenous, we can apply solvers searching for rational solutions. In order to decide if there is a non-negative rational solution and to find such a solution in the positive case, the ellipsoid method by Khachiyan [14] can be used. The runtime of this algorithm is polynomial in the size of the inequation system. Since there are at most $|L(\sigma)| \cdot |T|$ wrong continuations, the time complexity for computing the final net is polynomial in the size of the input event log σ. Although the method of Khachiyan yields an algorithm to solve the process mining problem in polynomial time, usually a better choice is the classical Simplex algorithm or variants of the Simplex algorithm [20]. While the Simplex algorithm is exponential in the worst case, probabilistic and experimental results [14] show that the Simplex algorithm has a significant faster average runtime than the algorithm of Khachiyan. The standard procedure to calculate a starting edge with the Simplex algorithm is a natural approach to decide, if there is a non-negative integer solution of the linear inequation system and to find such solution in the positive case. But it makes also sense to use the whole

Simplex method including a linear objective function. The choice of a reasonable objective function for the Simplex solver is a parameter of the algorithm to improve the results, e.g. a function minimizing the arc weights and the initial markings of the separating feasible places. Moreover, there are several variants of the Simplex algorithm that can improve the runtime of the mining algorithm [20]. For example the inequation systems for the wrong continuations only differ in the last inequation $\mathbf{b}_{wt} \cdot \mathbf{r} < \mathbf{0}$. This enables the efficient application of incremental Simplex methods.

Independently from the choice of the solver, certain separating feasible places may separate more than one wrong continuation. For not yet considered wrong continuations, that are prohibited by feasible places already added to the constructed net, we do not have to calculate a separating feasible place. Therefore we choose a certain ordering of the wrong continuations. We first add a separating feasible place for the first wrong continuation (if such place exists). Then we only add a separating feasible place for the second wrong continuation, if it is not prohibited by an already added feasible places, and so on. This way we achieve, that in the resulting net, various wrong continuations are prohibited by the same separating feasible place. The chosen ordering of the wrong continuations can be used as a parameter to positively adjust the algorithm. In particular, given a fixed solver, there always exists an ordering of the wrong continuations, such that the net has no redundant places. But in general the net may still include redundant places. Again easily identifiable redundant places are finally deleted from the computed net. The resulting process mining algorithm, called method 2, is shown in [3].

To calculate a net from the log of the running example with method 2, we consider the length- plus-lexicographic order of the 45 wrong continuations: b, c, d, e, aa, ae, aba, abc, abd, abe, ... To compute a separating feasible place for a given wrong continuation, we use the standard Simplex algorithm. We choose an objective function (for the Simplex algorithm) that minimizes all arc weights outgoing from the constructed place as well as the initial marking. Figure 3 shows the places resulting from the first five wrong continuations b, c, d, e and aa. In Figures we annotate the constructed separating feasible places with the wrong continuation, for which the place was calculated. The next wrong continuation ae leads the ae-place in Figure 4. Then aba is already prohibited by the aa-place and thus no additional place is computed. The next three wrong continuations abc, abd and abe lead to the respective separating feasible places in Figure 4. Then all remaining 35 wrong continuations are prohibited by one of the already

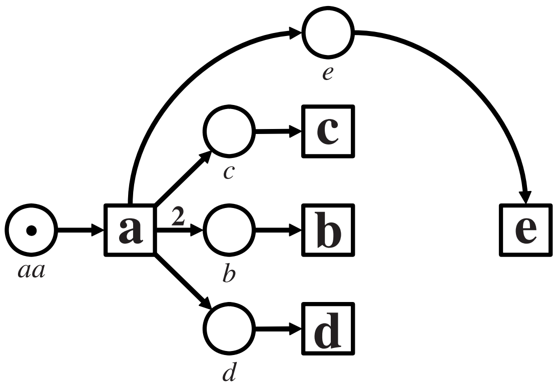

Fig. 3. First places computed with method 2

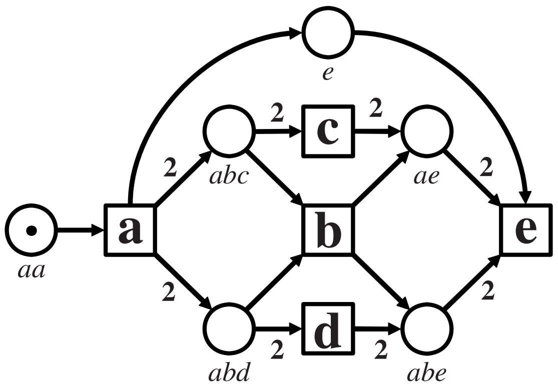

Fig. 4. Final net constructed with method 2

calculated nine feasible places. The b-, c-, and d-place from Figure 3 are finally deleted in Figure 4 as easily identifiable redundant places. Consequently the net in Figure 4 with six places results from Method 2 (only the e-place is still redundant).

The main advantage of method 2 is that the number of added places is bounded by $|L(\sigma)| \cdot |T|$ and that in most practical cases it is a lot smaller. Usually the resulting net is small and concise. The calculation of the net is efficient. There exists a polynomial time algorithm. Problematic is, that a good solution regarding (2) is not guaranteed, i.e. there may be intricate examples leading to a bad solution of the process mining problem. In [3] we show an example, where the constructed net is not optimal regarding (2), but this example was really hard to find. Therefore, in most cases the net should be an optimal solution. Moreover if the constructed net is not optimal, the respective example in [3] indicates that it is usually still a good solution of the process mining problem. Altogether the process model resulting from method 2 is a reasonable completion of the given probably incomplete log file. Although optimality regarding (2) is not guaranteed, the distinct advantages of method 2 concerning the runtime and the size of the calculated net altogether argue for method 2. But method 1 can still lead to valuable results, in particular if combined with some heuristics to decrease the number of places of the constructed net. Mainly, algorithms deleting redundant places are of interest.

3 Conclusion

The presented methods only considered p/t-nets as process models. To complete the outline of applying language based Petri net synthesis to process mining, we discuss alternative Petri net classes in this paragraph. In the example using method 1, we proposed to omit loops to simplify the constructed net. Leaving loops from p/t-nets in general, leads to the simpler class of pure nets. The process mining approach can analogously be developed for this net class. The inequation systems get simpler, in particular the number of variables is halved. Therefore the process mining approach gets more efficient for pure nets, but the modelling power is restricted in contrast to p/t-nets. Typical workflow Petri nets often have unweighted arcs. To construct such nets from a log with the presented methods, one simply has to add additional inequations ensuring arc weights smaller or equal than one. A problem is that the resulting systems are inhomogeneous. Method 1 is not applicable in this case (adaptions are still possible). Method 2 is still useable, but the linear programming techniques to find separating feasible places become less efficient [14]. A popular net class with unweighted arcs are elementary nets. In elementary nets the number of tokens in a place is bounded by one. This leads to additional inhomogeneous inequations ensuring this property. Note that the total number of possible places is finite in the case of elementary nets. Thus also the set of feasible places is finite leading to improvements of method 1. So far our considerations were based on the regions definition in [6,1]. There exists one further synthesis approach based on regions of languages [10], which we discuss and compare in [3].

The big advantage of the presented process mining approaches based on regions of languages is that they lead to reliable results. Other process mining algorithms are often more or less heuristic and their applicability is shown only with experimental results. We showed theoretical results that justify that the presented methods lead to a good or even optimal solution regarding (2), while (1) is guaranteed. A problem of the algorithms may be the required time and space consumption as well as the size of the resulting nets. The presented algorithms can be seen as a basis, that can be improved in several directions. Method 2 for computing separating feasible places is flexible w.r.t. the used solver and the chosen ordering of the wrong continuations. Varying the solver could improve time and space consumption, heuristics for fixing an appropriate ordering of the wrong continuations could lead to smaller nets. Both methods could be improved by additional approaches to find redundant places yielding smaller nets. For example, in this paper we used a simple special objective function in the simplex algorithm to rule out some redundant places. To develop such approaches, experimental results and thus an implementation of the algorithms is necessary.

References

1. Badouel, E., Bernardinello, L., Darondeau, P.: Polynomial algorithms for the synthesis of bounded nets. In: Mosses, P.D., Schwartzbach, M.I., Nielsen, M. (eds.) CAAP 1995, FASE 1995, and TAPSOFT 1995. LNCS, vol. 915, pp. 364–378. Springer, Heidelberg (1995)
2. Badouel, E., Darondeau, P.: Theory of regions. In: Reisig, W., Rozenberg, G. (eds.) Lectures on Petri Nets I: Basic Models. LNCS, vol. 1491, pp. 529–586. Springer, Heidelberg (1998)
3. Bergenthum, R., Desel, J., Lorenz, R., Mauser, S.: Technical report: Process mining based on regions of languages (2007), `http://www.informatik.ku-eichstaett.de/mitarbeiter/lorenz/techreports/bpm2007.pdf`
4. Cook, J.E., Wolf, A.L.: Discovering models of software processes from event-based data. ACM Trans. Softw. Eng. Methodol. 7(3), 215–249 (1998)
5. Cortadella, J., Kishinevsky, M., Kondratyev, A., Lavagno, L., Yakovlev, A.: Petrify: A tool for manipulating concurrent specifications and synthesis of asynchronous controllers. IEICE Trans. of Informations and Systems E80-D(3), 315–325 (1997)
6. Darondeau, P.: Deriving unbounded petri nets from formal languages. In: Sangiorgi, D., de Simone, R. (eds.) CONCUR 1998. LNCS, vol. 1466, pp. 533–548. Springer, Heidelberg (1998)
7. de Medeiros, A.K.A., van der Aalst, W.M.P., Weijters, A.J.M.M.: Workflow mining: Current status and future directions. In: Meersman, R., Tari, Z., Schmidt, D.C. (eds.) CoopIS 2003, DOA 2003, and ODBASE 2003. LNCS, vol. 2888, pp. 389–406. Springer, Heidelberg (2003)
8. Ehrenfeucht, A., Rozenberg, G.: Partial (set) 2-structures. part i: Basic notions and the representation problem. Acta Inf. 27(4), 315–342 (1989)
9. Ehrenfeucht, A., Rozenberg, G.: Partial (set) 2-structures. part ii: State spaces of concurrent systems. Acta Inf. 27(4), 343–368 (1989)
10. Lorenz, R., Bergenthum, R., Mauser, S., Desel, J.: Synthesis of petri nets from finite partial languages. In: Proceedings of ACSD 2007 (2007)
11. Motzkin, T.: Beiträge zur Theorie der linearen Ungleichungen. PhD thesis, Jerusalem (1936)
12. Parekh, R., Honavar, V.: Automata induction, grammar inference, and language acquisition. In: Dale, R., Moisl, H., Somers, H. (eds.) Handbook of Natural Language Processing, Marcel Dekker, New York (2000)
13. Process mining group eindhoven technical university: Prom-homepage, `http://is.tm.tue.nl/cgunther/dev/prom/`
14. Schrijver, A.: Theory of Linear and Integer Programming. Wiley, Chichester (1986)
15. van der Aalst, W., Rubin, V., van Dongen, B., Kindler, E., Guenther, C.: Process mining: A two-step approach using transition systems and regions. Technical Report BPM Center Report BPM-06-30, Department of Technology Management, Eindhoven University of Technology (2006)

16. van der Aalst, W., van Hee, K.: Workflow Management: Models, Methods, and Systems. MIT Press, Cambridge, Massachsetts (2002)
17. van der Aalst, W., Weijters, A., Verbeek, H.W., et al.: Process mining: research tools application, `http://www.processmining.org`
18. van der Aalst, W.M.P., van Dongen, B.F., Herbst, J., Maruster, L., Schimm, G., Weijters, A.J.M.M.: Workflow mining: A survey of issues and approaches. Data Knowl. Eng. 47(2), 237–267 (2003)
19. van Dongen, B., Busi, N., Pinna, G., van der Aalst, W.: An iterative algorithm for applying the theory of regions in process mining. Technical Report Beta rapport 195, Department of Technology Management, Eindhoven University of Technology (2007)
20. Vanderbei, R.J.: Linear Programming: Foundations and Extensions. Kluwer Academic Publishers, Dordrecht (1996)