Efficient Comparison of Process Models using Tabu Search Algorithm*

A.V. Skobtsov[1] and A.A. Kalenkova[1]

[1]National Research University Higher School of Economics, Moscow, Russia
{akalenkova@,avskobtsov@edu.}hse.ru

Abstract. Companies from various domains record their operational behavior in a form of event logs. These event logs can be analyzed and relevant process models representing the real companies' behavior can be discovered. One of the main advantages of the process discovery methods is that they commonly produce models in a form of graphs which can be easily visualized giving an intuitive view of the executed processes. Moreover, the graph-based representation opens new challenging perspectives for the application of graph comparison methods to find and explicitly visualize differences between discovered process models (representing real behavior) and reference process models (representing expected behavior). Another important area where graph comparison algorithms can be used is the recognition of process modeling patterns. Unfortunately, exact graph comparison algorithms are computationally expensive. In this paper, we adapt an inexact tabu search algorithm to find differences between BPMN (Business Process Model and Notation) models. The tabu search and greedy algorithms were implemented within the BPMNDiffViz tool and were tested on BPMN models discovered from synthetic and real-life event logs. It was experimentally shown that inexact tabu search algorithm allows to find a solution which is close to the optimal in most of the cases. At the same, its computational complexity is significantly lower than the complexity of the exact A^* search algorithm investigated earlier.

Keywords: graph edit distance, process mining, BPMN (Business Process Models and Notation), tabu search, conformance checking.

1 Introduction

Information systems automating processes in various fields, including medical care, education, e-government, banking, write history of their executions to event logs. Process mining techniques allow us to discover process models generalizing systems' behavior presented in these event logs [3].

To understand the meaning of the discovered process models, their structure can be additionally analyzed. For example, process modeling patterns such as

* Copyright © 2019 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0)

sequence, *choice*, *parallel execution*, *loop*, and other, more complicated structures can be identified automatically. Besides that, discovered process models can be compared to reference models explicitly showing deviations between real and expected process behavior.

In papers [1, 2], it was suggest to use a so-called *graph edit distance* to compare process models discovered from event logs. The graph edit distance shows the degree of model difference/similarity. More precisely, it is defined as a minimal number of elementary steps (node insertion/deletion, edge insertion/deletion, label editing) needed to transform one graph another. Methods for finding minimal graph edit distance can be applied to process models represented in a form of labeled oriented graphs with node and edge types. The exact A* search algorithm [4] was implemented in a tool [1] for the comparison of BPMN (Business Process Model and Notation) models [5]. This tool was used for the comparison of BPMN models discovered from event logs of e-government services [2]. To reduce the computational complexity of the model comparison technique, process-specific heuristics were used [2]. Nevertheless, it was still not possible to compare large process models extracted from event data. This can be explained by the fact that the problem of finding minimal graph edit distance is know to be NP-complete [6].

In this paper, we study the possibility of finding minimal-graph edit distance by applying an inexact tabu search algorithm [7]. The advantage of this algorithm is that the total number of iterations is bounded by a constant predefined number. A tabu search method for finding minimal-graph edit distance between graphs was proposed in [8]. This method was focused on the application in the computer vision field. In contrast to [8], we will compare process-specific models discovered from event data.

To estimate time and accuracy characteristics of the proposed algorithm, we will compare process models discovered from event logs using Alpha [9] and Inductive mining algorithms [10]. For the conversion of discovered process models to BPMN standard we will use transformation rules proposed in [11].

The paper is organized as follows. Section 2 presents main notions. Section 3 describes an exact graph matching technique based on A* search algorithm. An algorithm for finding graph edit distances between BPMN models using tabu search technique is presented in Section 4. Section 5 contains experiment results. And finally, Section 6 concludes the paper.

2 Preliminaries

In this section, we will define flat BPMN models, business process graphs and distances between them. These notions will be used through the paper.

Although there exists a wide range of models' types proposed by the Object Management Group (OMG) [5], we will consider flat BPMN models only. These models formalize the simple control flow. They can be discovered in two steps: (1) a low-level model (such as Petri net, causal net, or process tree) is synthesized from the event log; (2) this model is converted into a high-level model using algorithms presented in [11].

Flat BPMN models constructed from Petri nets, process trees or causal nets are represented by the following basic set of elements: *tasks*, *exclusive* and *parallel* gateways, *start* and *end events*, *control flows* (Fig. 1).

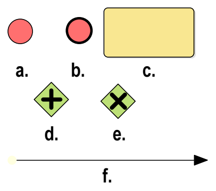

Fig. 1. Elements of flat BPMN models.

Start (Fig. 1 a.) and *end events* (Fig. 1 b.) denote the beginning and the termination of the process respectively. *Tasks* (Fig. 1 c.) model atomic process steps. *Parallel* (Fig. 1 d.) and *exclusive gateways* (Fig. 1 e.) are used to model process branches which are executed in parallel or are mutually exclusive.

An example of a BPMN model which presents a simple booking procedure is presented in Fig. 2. Firstly, the user registers, then books a flight or a hotel (these activities are performed in parallel and each of them can be skipped), and finally, pays.

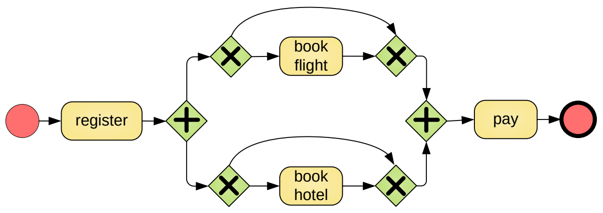

Fig. 2. A BPMN model describing a simple booking procedure.

Flat BPMN models can be considered as oriented graphs with node and edge types. We will call them *business process graphs*. Let us give their formal definition.

Business process graph is a tuple $G = (N, E, t, l)$, where N – is a set of nodes, $E \subseteq (N \times N)$ – a set of edges, $t : N \to T$, where $T = \{$*start event*, *end event*, *task*, *exclusive gateway*, *parallel gateway*$\}$ – is a function which defines node types, $l : N \to L$ – a function which defines labels, and L – is a set of labels.

Let us consider two business process graphs: $G_k = (N_k, E_k, t_k, l_k)$, $k = 1, 2$. The *distance* between these graphs is quantified by constructing an *edit relation* $R \subseteq (N_1 \cup E_1 \cup \{\epsilon, \delta\}) \times (N_2 \cup E_2 \cup \{\epsilon, \delta\})$, where $\epsilon, \delta \notin N_1 \cup N_2 \cup E_1 \cup E_2$, such that the following conditions hold:

1. for each element $i_1 \in N_1 \cup E_1$ $(i_2 \in N_2 \cup E_2)$ there exists one and only one element $i_2 \in N_2 \cup E_2 \cup \{\epsilon, \delta\}$ $(i_1 \in N_1 \cup E_1 \cup \{\epsilon, \delta\})$, such that $(i_1, i_2) \in R$;
2. if $i_1 \in \{\epsilon, \delta\}$ $(i_2 \in \{\epsilon, \delta\})$ and $(i_1, i_2) \in R$, then $i_2 \notin \{\epsilon, \delta\}$ $(i_1 \notin \{\epsilon, \delta\})$;
3. if $i_1 \in N_1$ and $(i_1, i_2) \in R$, then $i_2 \in N_2 \cup \{\epsilon, \delta\}$, and if additionally $i_2 \in N_2$, then the node types coincide, i.e., $t_1(i_1) = t_2(i_2)$;
4. if $i_1 \in E_1$ and $(i_1, i_2) \in R$, then $i_2 \in E_2 \cup \{\epsilon, \delta\}$;
5. for any $(i_1, i_1') \in E_1$, $(i_2, i_2') \in E_2$ the condition $((i_1, i_1'), (i_2, i_2')) \in R$ holds iff $(i_1, i_2) \in R$ and $(i_1', i_2') \in R$.

If for an element i holds that $(i, \epsilon) \in R$ ($(\epsilon, i) \in R$), then we say that i is *deleted* (*inserted*). If $(i, \delta) \in R$ ($(\delta, i) \in R$), then we say that there is no corresponding element for i (this will be used to represent intermediate comparison results).

Let us compare two business process graphs $G_k = (N_k, E_k, t_k, l_k)$, $k = 1, 2$ by constructing an edit relation R. For each $r = (i_1, i_2) \in R$ the cost will be calculated as follows:

$$cost(r) = \begin{cases} lev(l_1(i_1), l(i_2)) \cdot c_{lev}, & i_1 \in N_1, i_2 \in N_2, \\ 0, & i_1 \in E_1, i_2 \in E_2, \\ c_{delete}, & i_2 = \epsilon, \\ c_{insert}, & i_1 = \epsilon, \\ 0, & i_1 = \delta \vee i_2 = \delta. \end{cases}$$

Function *lev* calculates Levenshtein distance [12] between two labels, c_{lev} – is a special coefficient; c_{delete} and c_{insert} denote costs of deletion and insertion operations respectively.

The total cost of the edit relation R is defined as a sum of costs of all pairs which belong to this relation: $cost(R) = \sum_{r \in R} cost(r)$.

The minimal edit relation between two business process graphs is an edit relation with minimal cost, such that it does not contain pairs with δ (correspondences are defined for all elements). The cost of this relation is called *minimal graph edit distance.*

3 An Exact Algorithm for Finding Minimal Graph Edit Distance

A description of an exact algorithm for finding minimal graph edit distance is presented in this section (Algorithm 1). This algorithm takes a minimal graph edit relation from the queue at each step. Then from this edit relation novel edit relations are generated by applying *expand* function matching a node which has no correspondence yet (it corresponds to δ) to all possible nodes of the other model which have the same type and have no correspondence either (they correspond to δ) or to the ϵ element. When calculating new costs incident edges of this node are also considered. The algorithm stops when a minimal cost relation does not contain pairs with δ, i.e, there are correspondences for all elements. The cost of this edit relation is a minimal graph edit distance between these graphs.

Fig. 3 presents a results of comparison of business process graphs modeling a booking procedure. The business process graph presented in Fig. 3 a. models the booking procedure which was presented before (Fig. 2). The model shown in Fig. 3 b. assumes that there can be a cancellation. To transform the first business process graph (Fig. 3 a.) to the second (Fig. 3 b.) two edges are to be deleted and a subprocess corresponding to the cancellation should be added.

Although the structure of models is different, they have the same underlying booking procedure.

```
Data: $G_1 = (N_1, E_1, t_1, l_1)$ and $G_2 = (N_2, E_2, t_2, l_2)$ – business process
      graphs;
Result: minimal graph edit relation between $G_1$ and $G_2$;
\\$R_{init}$ – start edit relation;
$R_{init} \leftarrow \{\}$;
for $i_1 \in N_1 \cup E_1$ do
    $R_{init} \leftarrow R_{init} \cup \{(i_1, \delta)\}$;
end
for $i_2 \in N_2 \cup E_2$ do
    $R_{init} \leftarrow R_{init} \cup \{(\delta, i_2)\}$;
end
\\$Q$ - a sorted queue of edit relations;
$Q \leftarrow \langle R_{init} \rangle$;
while true do
    \\select edit relation with a minimal cost;
    $R_{min} \leftarrow takeMinCostRelation(Q)$;
    if ($R_{min}$ contains pairs with $\delta$) then
        $i \leftarrow takeNodeRelatedtoDelta(R_{min})$;
        $Q.remove(R_{min})$;
        \\add all possible pairs for $i$ to $R_{min}$;
        $Q.add(expand(R_{min}, i))$;
    else
        return $cost(R_{min})$;
    end
end
```

Algorithm 1: Finding minimal graph edit distance between business process graphs.

In order to make the algorithm more efficient, a special *heuristic function* can be used. It is defined on a set of possible relations as follows:

$$H(R) = \sum\nolimits_{t \in T} \begin{cases} (|N_1^t| - |N_2^t|) \cdot c_{delete}, & |N_1^t| \geq |N_2^t|, \\ (|N_2^t| - |N_1^t|) \cdot c_{insert}, & |N_2^t| > |N_1^t|. \end{cases}$$

$N_1^t \subseteq N_1$ and $N_2^t \subseteq N_2$ denote sets of model nodes of type t with no correspondences currently defined. Then the total cost is calculated as: $cost(R) = \sum_{r \in R} cost(r) + H(R)$. Indeed, using a heuristic function, one can predict the number of nodes for which no corresponding nodes will be found in another graph, so they are guaranteed to be removed or added. Let us consider an edit relation R' obtained from R by matching unmatched elements, such that for all $(i_1, i_2) \in R'$ it holds that $i_1 \neq \delta$ and $i_2 \neq \delta$, i.e., all elements in R' are matched. Then, $cost(R') = cost(R) + \sum_{r \in \overline{R}} cost(r)$, where $\overline{R} = \{(i_1, i_2) \in R' | (i_1, \delta) \in R \vee (\delta, i_2) \in R\}$. Let us consider nodes of type $t \in T$. Since we construct one-to-one relation, the number of elements which are to be deleted is at least $|N_1^t| - |N_2^t|$ (if

$|N_1^t| \geq |N_2^t|$), or the number of nodes which are to be added is at least $|N_2^t| - |N_1^t|$ (if $|N_2^t| > |N_1^t|$). Thus, $\sum_{r \in \overline{R}} cost(r) \geq H(R)$, and the heuristic function does not overestimate the final result.

The approach based on the use of a heuristic function is also known as A^* search algorithm [4].

Another heuristics that can be used is that we can first consider those nodes that are connected with nodes for which correspondences are already defined. This heuristics allows us to efficiently find a solution when comparing similar process models.

The proposed heuristics were implemented in [1] and were tested on business processes graphs synthesized from event logs of real-life information systems [2]. Although these heuristics make the algorithm of comparing business process graphs work faster, the complexity of the problem associated with its NP-completeness remains. This necessitates the development of imprecise but fast methods that will allow us finding near minimum graph edit distances between large process models.

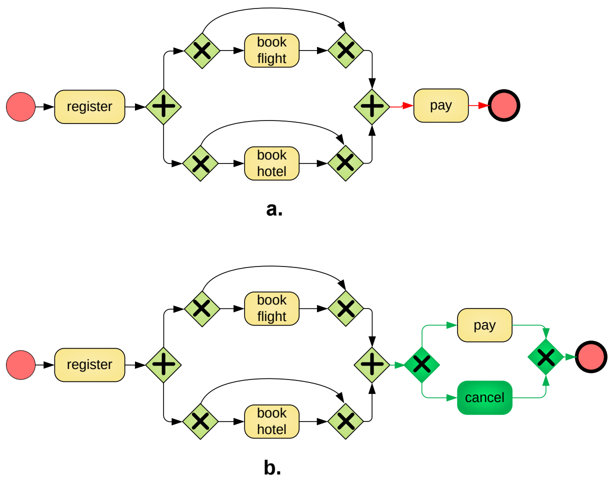

Fig. 3. The result of comparison of business process graphs modeling a booking procedure.

4 Application of Tabu Search Algorithm for the Comparison of Business Process Graphs

This section contains a description of a so-called tabu search algorithm adapted to solve the problem of finding minimal edit distance between graphs (Algorithm 2). The tabu search algorithm moves through a graph of solutions choosing the best neighboring solution at each time. To avoid looping, a special *tabuList* is used. This list contains solutions which were already considered. At the same time, it may have a limited length (*maxTabuListSize*) in order to provide a better performance. Another parameter of the algorithm is *maxExpansions*, it limits the overall number of steps. Note that all of the aforementioned solutions are both belong to R and there are no unmatched pairs in any of them, i.e. there is no pair with δ in any solution.

Firstly, the current solution R_{cur} is initialized by applying a greedy algorithm. Then, it is added to *tabuList*, and neighboring solutions are generated. After that, the neighboring solutions which belong to *tabuList* are filtered out and the one with a minimal cost is selected. If its cost is lower than the global minimal cost, then the global minimal cost is updated. If *tabuList* is full, the first (the oldest) solution is removed. Finally, after the fixed number of steps (*maxExpansions*) is completed or if there are no neighboring solutions due to filtering out, the current global minimal cost is returned. Note that the algorithm can also return the solution corresponding to this minimal cost.

```
Data: $G_1 = (N_1, E_1, t_1, l_1)$ and $G_2 = (N_2, E_2, t_2, l_2)$ – business process
      graphs; $maxTabuListSize$ – the maximal length of the tabu list;
      $maxExpansions$ – the maximal number of state expansions;
Result: graph edit distance between $G_1$ and $G_2$;
\\initialize $R_{cur}$ – edit relation;
$R_{cur} \leftarrow R_{greedy}$;
$minCost \leftarrow cost(R_{cur})$;
$tabuList.add(R_{cur})$;
foreach $expansion \in \{1, \cdots, maxExpansions\}$ do
    $oneStepVariants \leftarrow generateOneStepVariants(R_{cur})$;
    foreach $variant \in oneStepVariants$ do
        if $variant \in tabuList$ then
            $oneStepVariants.remove(variant)$;
        end
    end
    if $oneStepVariants = \{\}$ then
        break;
    end
    $R_{cur} \leftarrow takeMinCostRelation(oneStepVariants)$;
    if $minCost > cost(R_{cur})$ then
        $minCost \leftarrow cost(R_{cur})$;
    end
    if $tabuList.size() = maxTabuListSize$ then
        $tabuList.removeFirst()$;
    end
    $tabuList.add(R_{cur})$;
end
return $minCost$;
```

Algorithm 2: Finding a minimal edit distance between business process graphs by applying tabu search algorithm.

Now let us describe the *generateOneStepVariants* procedure for constructing all possible neighboring solutions of the current edit relation R. The neighboring solutions are generated by applying one of the following substitution rules to one of the pairs belonging to R:

- $(n_1, n_2) \rightarrow (n_1, \epsilon), (\epsilon, n_2)$,
- $(n_1, \epsilon), (\epsilon, n_2) \rightarrow (n_1, n_2)$,
- $(n_1, \epsilon), (n'_1, n_2) \rightarrow (n_1, n_2), (n'_1, \epsilon)$,
- $(n_1, n_2), (\epsilon, n'_2) \rightarrow (n_1, n'_2), (\epsilon, n_2)$,
- $(n_1, n_2), (n'_1, n'_2) \rightarrow (n_1, n'_2), (\epsilon, n_2), (n'_1, \epsilon)$,
- $(n_1, n_2), (n'_1, n'_2) \rightarrow (n_1, \epsilon), (n'_1, n_2), (\epsilon, n'_2)$.

Note that the pairs specifying relations between edges will be defined in accordance with the definition of edit relation. Also note that we match elements with the same type only.

To initialize the current edit relation we will use greedy algorithm, which works as A^* (Algorithm 1) with the queue Q of size 1. As well as A^*, it starts with an empty relation and then adds pairs to this relation at each step selecting a relation with a minimal cost. It stops when there are no pairs with δ. In contrast to A^*, it has a linear computational complexity. In the next section, we will evaluate accuracy and performance characteristics of A^*, tabu and greedy algorithms by comparing BPMN models discovered from event logs.

5 Experimental results

This section contains results of experiments for finding minimal edit distances between BPMN models discovered by different algorithms [9, 10] from the same sets of artificial [1] and real-life event logs from e-government services, and online ticketing domains.

All the deletion and insertion costs were set to 1 during the experiments, i.e., $c_{delete} = c_{insert} = 1$, the Levenshtein distance coefficient was defined as $c_{lev} = 10$. For the tabu search the parameters were set as: $maxExpansions = 100$, $maxTabuListSize = 100$. All experiments were carried out on Asus ZenBook Pro, with the following characteristics: Intel i7-8750H 2.20 GHz processor, 16GB RAM.

Fig. 4 shows execution times (Fig. 4 a.) as well as minimal costs (Fig. 4 b.) obtained by A^*, tabu and greedy algorithms for BPMN modes discovered from artificial logs using different (Inductive and Alpha miner) algorithms.

Similarly, BPMN models discovered from real-life event logs of e-government services and online ticketing systems using different process mining algorithms were compared using A^*, tabu and greedy techniques (Fig. 5). The size of models was varied by filtering out different portions of infrequent behavior using the *Filter Log using Simple Heuristics* ProM 6.8 plugin. After filtering out each of the event logs was processed via *Alpha Miner* and *Mine Petri net with Inductive Miner* plugins with default parameters resulting into Petri nets. Finally, these Petri nets were converted to BPMN models and compared.

[1] http://www.processmining.org/event_logs_and_models_used_in_book.

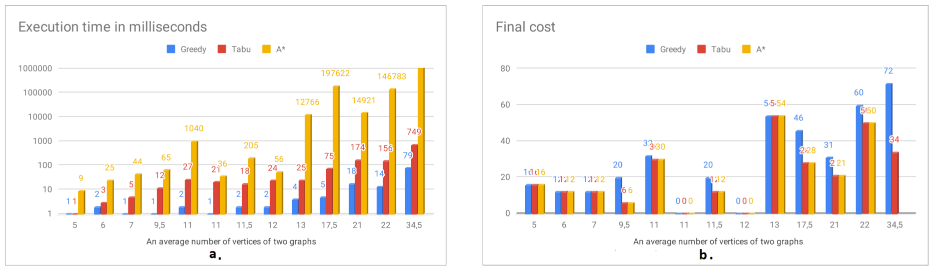

Fig. 4. Characteristics of the algorithms comparing BPMN models discovered from artificial event logs.

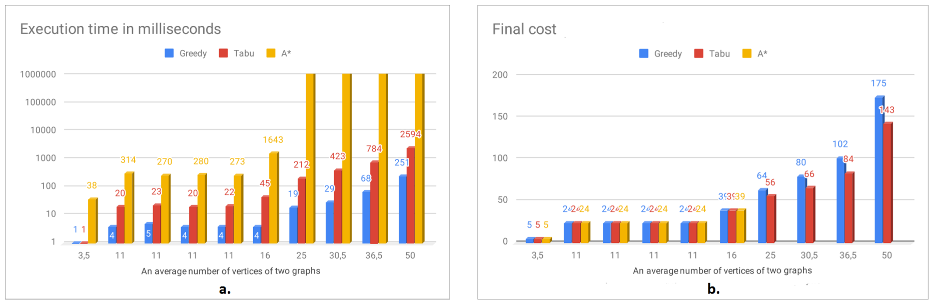

Fig. 5. Comparison of BPMN models discovered by Alpha [9] and Inductive [10] mining algorithms.

And finally, BPMN models discovered by the Inductive miner algorithm corresponding to different parts of event logs were compared. In this case, random 1000 traces were selected from the initial log using the *Extract sample of traces (Random)* plugin twice, then processed using the Inductive miner algorithm only. The results of this comparison are presented in Fig.6.

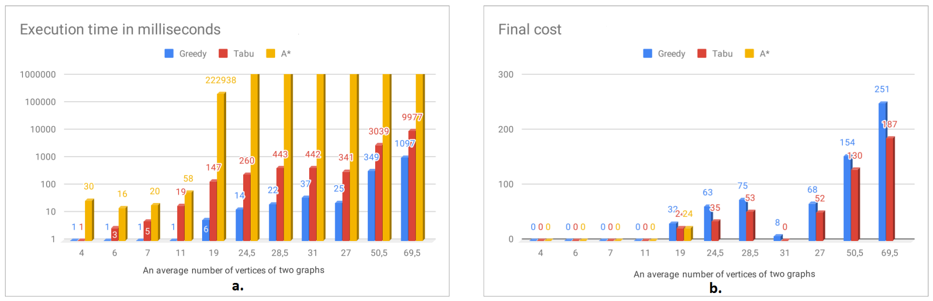

Fig. 6. Comparison of BPMN models constructed by Inductive [10] algorithm from different random parts of real-life event logs.

As it follows from the result presented in Fig. 4, 5, 6, the execution time of the exact A^* exceeds 30 minutes for large models. In such cases, it is more feasible to use tabu search, which is rather fast (as shown by the experiments) and produces optimal solutions in most of the experimental cases presented in Fig. 4, 5, 6.

6 Conclusion

Algorithms for the analysis and discovery of process models from information systems' event logs (process mining) are widely used. There are many commercial tools for the discovery of process models based on the event logs analysis. With the development of the theory of process mining the task of model to model comparison becomes extremely important, as it not only helps to find differences between expected and real behavior, but also to visualize the result (this characteristic of process analysis algorithms is especially appreciated by end users). Due to the fact that in general the problem of graph models comparison is NP-complete, heuristic methods for the comparison of process models are particularly valuable. In this paper, we adapted a greedy and tabu search algorithms for the problem of matching process models extracted from event logs. These algorithms were implemented and tested on BPMN models. Both the accuracy and the time characteristics of the algorithms were evaluated. It was demonstrated that the tabu search algorithm helps to significantly improve the time, while producing optimal results in most of the cases.

Acknowledgment. This work was funded by the President Grant MK-4188.2018.9.

References

1. Ivanov, S.Y., Kalenkova, A.A., van der Aalst, W.M.P.: BPMNDiffViz: A Tool for BPMN Models Comparison. In: Proceedings of the BPM Demo Session 2015 Co-located with the 13th International Conference on Business Process Management (BPM 2015), Innsbruck, Austria, September 2, 2015. (2015) 35–39
2. Kalenkova, A.A., Ageev, A.A., Lomazova, I.A., van der Aalst, W.M.P.: E-Government Services: Comparing Real and Expected User Behavior. In Teniente, E., Weidlich, M., eds.: Business Process Management Workshops, Cham, Springer International Publishing (2018) 484–496
3. van der Aalst, W.: Process Mining: Data Science in Action. 2 edn. Springer (2016)
4. Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE transactions on Systems Science and Cybernetics **4**(2) (1968) 100–107
5. OMG: Business Process Model and Notation (BPMN). Object Management Group, formal/2013-12-09 (2013)
6. Garey, M.R., Johnson, D.S.: Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York, NY, USA (1990)

7. Glover, F.: Future paths for integer programming and links to artificial intelligence. Comput. Oper. Res. **13**(5) (May 1986) 533–549
8. Adamczewski, K., Lee, Y.S.K.M.: Discrete tabu search for graph matching. In: International Conference on Computer Vision. (2015)
9. van der Aalst, W.M.P., Weijters, A.J.M.M., Maruster, L.: Workflow mining: Discovering process models from event logs. IEEE Transactions on Knowledge and Data Engineering **16**(9) (2004) 1128–1142
10. Leemans, S., Fahland, D., van der Aalst, W.: Discovering Block-Structured Process Models from Incomplete Event Logs. In: 35th International Conference, PETRI NETS 2014. Volume 8489 of LNCS. Springer, Cham (2014) 91–110
11. Kalenkova, A., van der Aalst, W., Lomazova, I., Rubin, V.: Process mining using bpmn: relating event logs and process models. Software and Systems Modeling (2015) 1–30
12. Levenshtein, V.: Binary Codes Capable of Correcting Deletions, Insertions and Reversals. Soviet Physics Doklady **10** (1966) 707