Int. J. Appl. Math. Comput. Sci., 2010, Vol. 20, No. 3, 591–600
DOI: 10.2478/v10006-010-0044-0

NUMBERING ACTION VERTICES IN WORKFLOW GRAPHS

ZOLTÁN ÁDÁM MANN

Department of Computer Science and Information Theory
Budapest University of Technology and Economics, Magyar tudósok körútja 2., 1117 Budapest, Hungary
e-mail: zoltan.mann@gmail.com

AAM Management Information Consulting Ltd.
Váci út 76., 1133 Budapest, Hungary

Workflow graphs, consisting of actions, events, and logical switches, are used to model business processes. In order to easily identify the actions within a workflow graph, it is useful to number them in such a way that the numbering reflects the structure of the workflow. However, available tools offer only rudimental numbering schemes. In the paper, a set of natural requirements is defined that a logical numbering should fulfill. It is investigated under what conditions there is an appropriate numbering at all, when it is uniquely defined by the set of requirements, and when it can be computed efficiently. It is shown that for an important special class of workflow graphs, namely, structured workflow graphs, the answer to all these questions is affirmative. For general workflow graphs, a set of requirements is presented that can always be fulfilled, but the numbering is not necessarily unique. An algorithm based on a depth-first search can be used to compute an appropriate numbering efficiently.

Keywords: workflow graph, flowchart, event-driven process chain, numbering, depth-first search.

1. Introduction

For the graphical representation of business processes, several flavours of flowcharts are in use, with slight differences in syntax and semantics. In this paper, we will use the *standard workflow model* of Kiepuszewski *et al.* (2003), a quite general class of workflow graphs. In our experiments we used a specific modeling language, the so-called event-driven process chain (EPC) (Scheer *et al.*, 2005; van der Aalst, 1999).

Informally, a workflow graph is a sequence of *actions*. Each action may be assigned an *actor*, some *input documents*, some *output documents*, and further attributes (see Fig. 1). If the process is not merely consecutive but consists of parallel and/or alternative paths, these paths are connected through *logical switches* of type AND, OR, or XOR. Moreover, every path starts with an *event* specifying the precondition for that path. Figure 2 shows a simple example of a business process of a store selling computer peripherals.

The optimization of business processes involves a lot of discussions about the corresponding workflow graphs. Reasoning about a workflow graph is eased significantly if the actions are numbered in some logical order. Logical order means that the numbering reflects the structure of the workflow graph and the order in which humans usually read workflow graphs, just like in Fig. 2. Of course, the graph in the figure is small, but with bigger graphs spanning several pages a logical numbering greatly improves orientation. Therefore, it is common sense in the business process optimization community that such a numbering is necessary. Keeping the numbering consistent after addition and deletion of actions is tedious, and hence computer support would be welcome, especially since business processes are usually modeled with computer tools anyway.

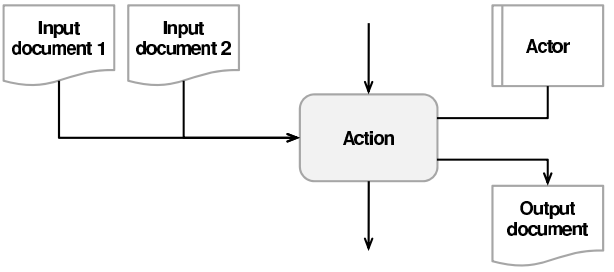


Fig. 1. Action and its attributes in a workflow graph.

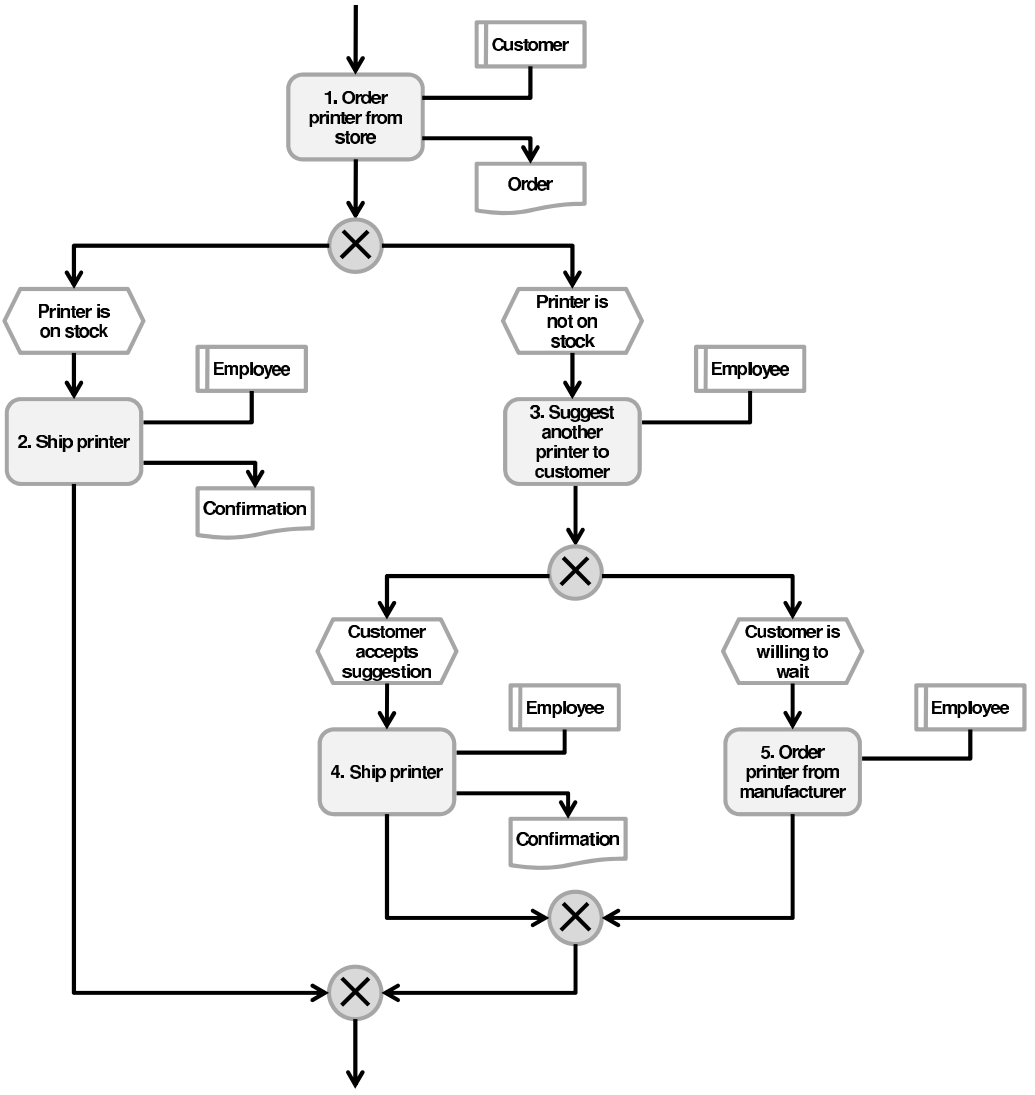


Fig. 2. Exemplary workflow graph.

However, available business process modeling tools offer only very rudimental support for numbering. For instance,

- ARIS (Scheer, 2000) offers no specific support for numbering but only allows the user to input the number as part of the action's name or as an attribute.
- Visio (Microsoft Corporation, 2010) offers an auto-numbering feature. However, this works by numbering the actions in the order in which they are created. Maintaining a logical numbering that would reflect the structure of the workflow graph is not supported.
- RFFlow (RFF Electronics, 2004) lets the user specify the number for each action. It has an automatic renumbering feature, which makes the numbers consecutive without changing their order. While this is certainly a useful feature, it is still the user's task to define the order of the nodes, as it is never changed by the program.

The reason for this lack of appropriate tool support is that it is by no means trivial to define precisely what a logical numbering should be like. Hence, in this paper, we start by defining a set of requirements that a logical numbering should fulfill. To do so, we take into account the structure of the graph as well as the way it is drawn in the plane, because these are the two most important factors for the human reader. After defining the set of requirements, we investigate if an appropriate numbering can always be found and if this numbering is uniquely defined by the set of requirements. Moreover, we investigate if the numbering can be computed efficiently. We will do this first for an important subset of workflow graphs, the so-called struc-

tured workflow graphs. As turns out, the answer to all these questions is positive in this case. In a second step, we investigate how much this can be transferred to general workflow graphs.

To our knowledge, the numbering of vertices in a workflow graph with the aim of improving readability has not been considered yet in the scientific literature. Of course, there are well-known basic algorithms for numbering vertices of a graph, e.g., topological order, the numberings generated by a depth-first search and a breadth-first search, a preorder/inorder/postorder search etc. In this paper, we will adapt a depth-first search to suit our particular goals. To our knowledge, this is the first work offering a mathematical approach to the problem of numbering vertices in a workflow graph with the aim of improving readability.

In some specific application domains, numbering problems somewhat similar to the problem covered in this paper have been studied. The most relevant example is in the field of workflow graph parsing. Recently, Vanhatalo and colleagues introduced the refined process structure tree to represent the structure of any workflow graph by decomposing it into a hierarchy of sub-workflows (Vanhatalo *et al*., 2009). While the process structure tree can be used for numbering purposes, it suits the needs of automatic processing of workflow graphs more than the needs of a human reader. In particular, the process structure tree does not take into account the geometry of the workflow graph's embedding into the plane, which is a vital clue for the human reader.

Another related work is the algorithm called M-propagation by Weber *et al.* (2008). M-propagation is part of a bigger algorithm, aiming at determining the consistency of workflow graphs that are annotated with semantic constraints. M-propagation incorporates a numbering with the aim of determining which pairs of actions can be active in parallel in a workflow graph. Again, this numbering is more for automatic processing than for human readers, and it does not take into account the geometry of the graph in the plane.

2. Preliminaries

For our purposes, a workflow graph can be modeled as follows:

Definition 1. A *workflow graph* is a directed graph $G = (V, E)$. The in-degree of vertex v will be denoted by $d_-(v)$, its out-degree by $d_+(v)$. In a workflow graph, there are the following types of vertices[1]:

- *Actions* with $d_-(v) = d_+(v) = 1$,
- *Splits* with $d_-(v) = 1$, $d_+(v) > 1$,
- *Joins* with $d_-(v) > 1$, $d_+(v) = 1$,
- A *start vertex* with $d_-(v) = 0$, $d_+(v) = 1$,
- An *end vertex* with $d_-(v) = 1$, $d_+(v) = 0$.

By denoting the set of actions with A, the set of splits with S, the set of joins with J, the start vertex with s and the end vertex with e, we have $V = A \cup S \cup J \cup \{s, e\}$, where the sets A, S, J, and $\{s, e\}$ are pairwise disjoint.

It is assumed that each vertex is reachable from s and e is reachable from each vertex.

The workflow graph is embedded in the plane, but the crossing of edges is allowed. Each vertex v has its corresponding coordinates $x(v)$ and $y(v)$. We will consider the graph as being acyclic and all edges as being directed downwards: $uv \in E \Rightarrow y(u) > y(v)$ (see the explanation below).

The restriction that the graph has a single start and end vertex is purely technical. It makes the presentation more readable, and it is not a real restriction. For example, a workflow graph with multiple start vertices can be extended with a new, single start vertex, which is connected through a new split to the old start vertices.

The other assumption that we made is that the graph should be acyclic. This might seem too restrictive, but actually it is not restrictive at all. The downward orientation of edges is a common convention in business process modelling. If there is a cycle of length k in a process, then this is usually depicted with $k-1$ edges going downwards, describing the „main flow", completed by one upward arrow indicating the repetitions. Given our aim to define a logical order of the actions, we can ignore the single upward arrow, because it does not contain any information that should be taken into account in the numbering. In other words, we do not restrict ourselves to acyclic workflow graphs. Rather, the convention that the main flow is given by downward arrows implies that—for the purposes of defining a logical numbering—upward arrows can be ignored, thus making the remaining workflow graph acyclic.

The embedding of the graph in the plane implies an order on the outgoing edges of a split and the incoming edges of a join.

Definition 2. If v is a split with outgoing edges $e_1, e_2, \ldots, e_k$, then we write $e_1 < e_2 < \ldots < e_k$ if e_1 is the leftmost, e_2 the second from the left, and so on (see Fig. 3), and similarly for incoming edges of joins.

Definition 3. A workflow graph is called *structured* if it can be constructed using the following rules:

1. If $S = J = \emptyset$, then the workflow graph is structured.

[1]The other vertex types (events, actors, input and output documents etc.) can be ignored because they are not relevant for the numbering. Likewise, it is indifferent if a split or join is of type AND, OR, or XOR.

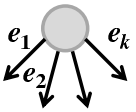
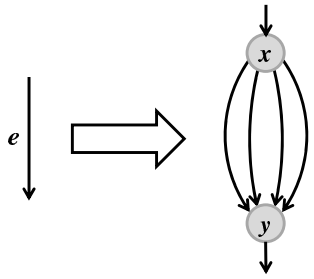


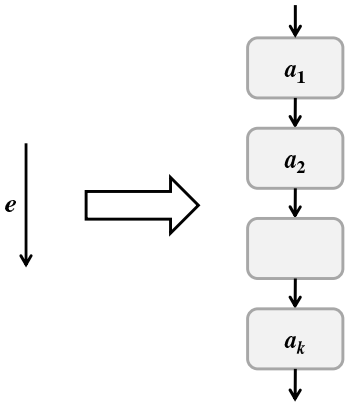
Fig. 3. Ordered set of edges going out from a split.

2. Let G be a structured workflow graph and $e \in E_G$ an edge in G. The workflow graph G' is constructed from G as follows: e is substituted by a split x and a join y such that the number of edges going out from x equals the number of edges coming to y and the first edge going out from x is the first edge that comes into y, the second edge going out from x is the second edge that comes into y etc. (see Fig. 4(a)). Then G' is also structured.

3. Let G be a structured workflow graph and $e \in E_G$ an edge in G. The workflow graph G' is constructed from G as follows: e is substituted by a chain of new actions $a_1, a_2, \ldots, a_k$ with edges $a_1a_2, \ldots, a_{k-1}a_k$ between them (see Fig. 4(b)). Then G' is also structured.



(a) Rule 2



(b) Rule 3

Fig. 4. Transformations for constructing structured workflow graphs.

As an example, the workflow graph of Fig. 2 is structured. However, not every workflow graph is structured. For instance, the workflow graph in Fig. 5 cannot be structured because it contains three splits but only two joins. Structured workflow graphs are analogous to `goto`-free computer programs, in which branching is realized by `then` and `else` instruction blocks of `if-then-else` statements. In contrast, a non-structured workflow graph is analogous to a program containing `goto` instructions. Just as `goto`-free programs are preferable over programs containing `goto` instructions, it is good practice to model processes by structured workflow graphs, and hence structured workflow graphs represent an important special case of workflow graphs.

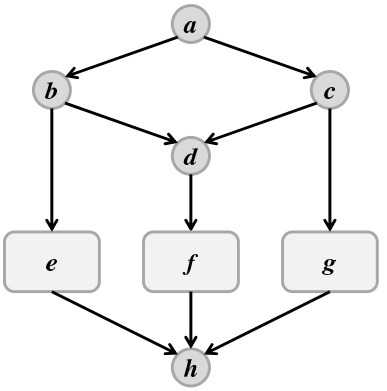


Fig. 5. Example of a non-structured workflow graph.

Definition 4. A *numbering* of a workflow graph is a bijective function $n : A \to \{1, 2, \ldots, |A|\}$, where A denotes the set of actions within the workflow graph and $|A|$ denotes the cardinality of the set A.

3. Numbering structured workflow graphs

A structured workflow graph is characterized by the corresponding split—join pairs and two or more branches between the split and the join of each pair. The order of outgoing edges of the split defines an order on the set of branches between a corresponding split–join pair.

Definition 5. Let x and y be a split and the corresponding join, respectively. The branches between them are denoted by $B_1, B_2, \ldots, B_k$, the edge between x and B_i is denoted by e_i (see Fig. 6). Then $B_i < B_j$ if $e_i < e_j$.

In this section, it is investigated what a logical numbering of a structured workflow graph should be like. The numbering should be in line with the order in which a workflow graph is usually read. Basically, this means going from top to bottom and from left to right. More formally, the numbering should fulfill the following requirements:

Requirement 1. If $a_1, a_2 \in A$ and there is a directed path from a_1 to a_2, then $n(a_1) < n(a_2)$.

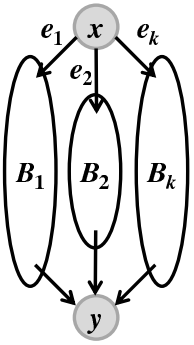


Fig. 6. Order of branches in a structured workflow graph.

Requirement 2. Let x and y be a split and the corresponding join, respectively. Let B_1 and B_2 be two branches between them with $B_1 < B_2$. Let $a_1 \in A \cap B_1$ and $a_2 \in A \cap B_2$ be two actions in the respective branches. Then $n(a_1) < n(a_2)$.

Please note how the requirements combine graph-theoretic and geometric notions to describe the required behavior.

Note also that Requirement 1 requires the numbering to be a topological order (although not on all vertices but only on the actions). Since the workflow graph is acyclic, Requirement 1 can be fulfilled. Moreover, it is well known that a depth-first search (DFS) can be used to construct a topological order in an acyclic graph (Cormen *et al*., 2001). With a suitable extension to accommodate the left-to-right rule, the DFS can be adapted to also fulfill Requirement 2. The result is shown as Algorithm 1.

It is worth mentioning that in Algorithm 1 the children of a vertex are visited intentionally *from right to left*. This is because the numbers are assigned in *decreasing* order.

Theorem 1. *For a structured workflow graph, Algorithm 1 results in a numbering that fulfills Requirements 1 and 2.*

Proof. It is clear that the result is indeed a numbering, i.e., each action gets a number between 1 and $|A|$ and each number is used exactly once.

It is well known that this numbering fulfills Requirement 1 (Cormen *et al*., 2001).

In order to prove that the numbering fulfills Requirement 2, consider the situation depicted in Fig. 6. Let y_i be the end vertex of edge e_i. When the algorithm visits x, none of the y_is can be visited because the workflow graph is structured. The algorithm first visits y_k. The call DFS(y_k) terminates only when all vertices in B_k have been visited. Since the workflow graph is structured, the vertices of the other branches are still not touched. Then

Algorithm 1 Tailored DFS algorithm to number structured workflow graphs

```
procedure DFS(x)
{
    visited(x):=true
    Let xy_1 < ... < xy_k be the edges going out of x
    for(i = k; i > 0; i − −)
        if not visited(y_i)
            DFS(y_i)
    if x ∈ A
    {
        n(x) := num
        num − −
    }
}

procedure main
{
    foreach vertex x
        visited(x):=false
    num:=|A|
    DFS(s)
}
```

comes the call DFS(y_{k-1}), which terminates when all vertices in B_{k-1} are visited, and so on. As a consequence, the actions in B_k get the highest numbers, the actions in B_{k-1} get lower numbers, the actions in B_{k-2} get even lower numbers, and so on. ■

Corollary 1. *In a structured workflow graph, there is always a numbering that fulfills Requirements 1 and 2.*

The next target is to investigate if the numbering is uniquely determined by the set of requirements. As we will see, for structured workflow graphs the answer is affirmative.

Definition 6. Let G be a structured workflow graph. The number of split–join pairs in G is denoted by $p(G)$.

Theorem 2. *For a structured workflow graph, there can only be one numbering that fulfills Requirements 1 and 2.*

Intuitively, the statement of the theorem is easy. If there is a directed path between two vertices, then Requirement 1 determines which of them should receive the smaller number. Otherwise, they are in different branches, and then Requirement 2 determines which of them should receive the smaller number. However, in a structured workflow graph, branchings can be nested into each other at arbitrary depth, and this complex structure makes it a bit more tricky to analyze the relation between two vertices. The following proof uses induction to handle this potentially nested structure.

Proof. We use induction according to $p(G)$.

If $p(G) = 0$, then the workflow graph consists of a single directed path. In this case, Requirement 1 guarantees the uniqueness of the numbering.

Let G be a structured workflow graph with $p(G) = t > 0$ and let us assume that the theorem is already proven for smaller values of p. Let x be the split nearest to s and let y be the corresponding join. Let the branches between x and y be denoted by $B_1, \ldots, B_k$. Let the part of the workflow graph between s and x be denoted by G_x, the part between y and e by G_y, as in Fig. 7. It is clear that $G_x, G_y, B_1, \ldots, B_k$ are all structured workflow graphs with a smaller p value. The set of actions in these workflow graphs will be denoted by $A_x, A_y, A_1, \ldots, A_k$, respectively, and let $A_B := A_1 \cup \ldots \cup A_k$.

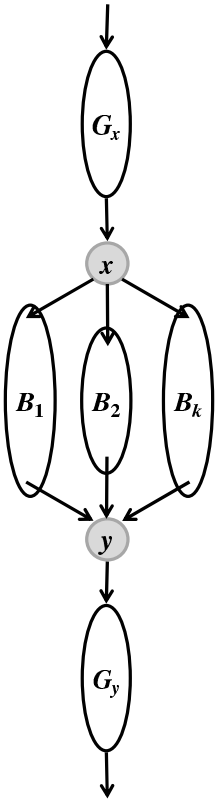


Fig. 7. Blocks in a structured workflow graph.

Because of Requirement 1, any action in A_x must have a smaller number than any action in A_B, and any action in A_B must have a smaller number than any action in A_y. Hence, the numbers $1, \ldots, |A_x|$ must be used for A_x, the numbers $|A_x| + 1, \ldots, |A_x| + |A_B|$ must be used for A_B, and the numbers $|A_x| + |A_B| + 1, \ldots, |A|$ must be used for A_y.

Because of Requirement 2, any action in A_1 must have a smaller number than any action in A_2, any action in A_2 must have a smaller number than any action in A_3, etc. Hence, from the numbers to be used for A_B ($|A_x| + 1, \ldots, |A_x| + |A_B|$), the first $|A_1|$ numbers have to be used for A_1, the next $|A_2|$ numbers must be used for A_2, etc.

Thus, Requirements 1 and 2 determine uniquely the interval of numbers to be used for each of the workflow graphs $G_x, G_y, B_1, \ldots, B_k$. Since these are all structured workflow graphs for which the theorem is already proven, it follows that the number of each action within each of these workflow graphs is uniquely determined. ■

To sum up the results for structured workflow graphs:

- There is always a numbering that fulfills the requirements.
- The numbering is unique.
- The numbering can be found efficiently.

4. Numbering general workflow graphs

The above results are based on the characteristics of structured workflow graphs, namely, the fact that split–join pairs define branches and there are no connections between distinct branches. In general workflow graphs, splits and joins are not in pairs and there are no defined independent branches. This invalidates for general workflow graphs not only the above results, but also the set of requirements, because Requirement 2 is based on the notion of branches. When considering real-life workflow graphs, this is also true: in contrast to structured workflow graphs where the logical order of the actions is clear, this is not always the case for non-structured workflow graphs. This situation is illustrated in Fig. 8, where it can be seen that it is not clear how the numbers of actions v and w should relate to each other. On the one hand, according to the order of the edges going out from y, $n(v) < n(w)$ would seem logical. On the other hand, there is a path from x to w starting with edge e_1 and a path from x to v starting with edge e_2, and since $e_1 < e_2$, $n(w) < n(v)$ should follow, which is a contradiction. Hence, the numbering problem is much more challenging for general workflow graphs than for structured workflow graphs.

In what follows, the goal is to transfer the ideas of Section 3 to general workflow graphs as much as possible. Requirement 1 can be transferred without problems. However, instead of Requirement 2, other requirements are needed to describe the left-to-right rule without using the notion of branches. Lacking the global structure of structured workflow graphs, this leads to a more fine-grained set of requirements, describing at a local level how the numbers assigned to adjacent vertices relate to each other. In other words, the requirements describe how numbers propagate from action to action—possibly through several splits and joins. For this, we will assign auxiliary

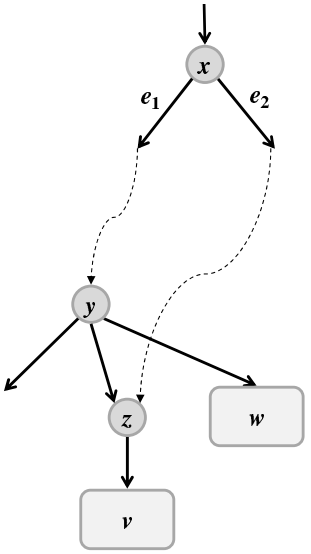


Fig. 8. Non-structured workflow graph, in which the logical order of actions is not obvious. Dashed lines indicate paths.

numbers to splits and joins as well. Informally, the number of a split or join is the last number used for an action until that point.

Definition 7. An *extended numbering* of a workflow graph is a function $n : V \to \{0, 1, 2, \ldots, |A|\}$, such that n is bijective between the subsets $A \subseteq V$ and $\{1, 2, \ldots, |A|\}$.

Now the proposed set of requirements (for an extended numbering of a general workflow graph) is as follows:

Requirement I. If $v_1, v_2 \in V$ and there is a directed path from v_1 to v_2, then $n(v_1) \leq n(v_2)$.

Requirement II. Let x be a split and let $xy_1 < xy_2 < \ldots < xy_k$ denote those outgoing edges whose end vertex is not a join. Then $n(y_1) \leq n(y_2) \leq \ldots \leq n(y_k)$.

Requirement III. Let $xy \in E$ be an edge. Then the following statements are true:

III(a): If $x \in A$ and $y \in A$, then $n(y) = n(x) + 1$.

III(b): If $x \in A$ and $y \in S$, then $n(y) = n(x)$.

III(c): If $x \in J$ and $y \in A$, then $n(y) = n(x) + 1$.

III(d): If $x \in J$ and $y \in S$, then $n(y) = n(x)$.

III(e): If $x \in S$, $y \in A$, and xy is the left-most of the edges going out of x, then $n(y) = n(x) + 1$.

III(f): If $x \in S$, $y \in S$, and xy is the left-most of the edges going out of x, then $n(y) = n(x)$.

III(g): If there holds $y \in J$ with incoming edges $x_1y, x_2y, \ldots, x_ky$ and $x_1, x_2, \ldots, x_k \notin S$, then $n(y) = \max\{n(x_i) : i = 1, 2, \ldots, k\}$.

In the following, we shall give some justification to this set of requirements.

Requirement I is very similar to the earlier Requirement 1. There are only two minor differences resulting from the fact that now we are looking for an extended numbering: Requirement I is valid for all vertices, not only for actions, and the numbers of two distinct vertices can be equal.

Requirement II is an adaptation of Requirement 2. Since, in the case of general workflow graphs, the notion of branches is not defined, we can only claim the left-to-right rule for the direct successors of a split. As for the indirect successors, Requirement III describes how the numbers must propagate along the edges. If the start vertex of an edge is not a split and its end vertex is not a join, then the numbers of the two vertices determine each other unambiguously (Requirements III(a–d)). If the start vertex of the edge is a split, then there is a similar relation between the split and its left-most child (Requirements III(e–f)). This is because the numbering must go on with the left-most child. Thus this vertex must receive the next number. The number of the other children of the split are not determined directly by the number of the split. The case when the end vertex is a join is described by Requirement III(g). There is a notable difference between the latter two cases: the split's number is related to the number of its left-most child, whereas the join's number is related to the maximum of its parents' numbers. In order to make the two more similar, we could call the left-most child of the split also the child with a minimum number, because of Requirement II. However, it is not true that the join's parent with a maximum number is the right-most one, not even for structured workflow graphs, as Fig. 9 shows.

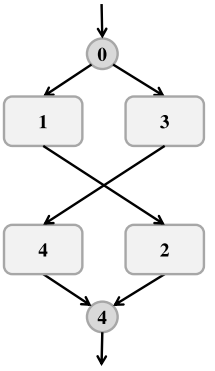


Fig. 9. Example of a join whose parent with a maximum number is not the right-most one.

Note also that in the case of an edge between a split and a join, special care is needed. As can be seen above,

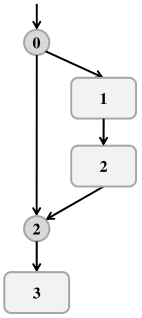


Fig. 10. Example showing why joins are excluded from Requirement II and Requirements III(e–f).

joins are excluded from Requirement II and there is nc rule like Requirements III(e) and III(f) for the case when $y \in J$. This is because, in the case of an edge from a split to a join, the number of the join is influenced by its other parents as well, not only by the split. Thus, imposing such rules on the join could lead to a contradiction. This is shown in Fig. 10, where the left-most child of the split is a join, and yet this join has a higher number than its sibling to the right and a much higher number than the split. This is why joins are excluded from Requirement II and there is no rule like Requirements III(e) and III(f) for the case when $y \in J$. Furthermore, it can also be seen that, in Requirement III(g), the case when one of the parents of a join is a split is excluded. The reason is that, in such a case, the number of the join is not necessarily determined by the numbers of its parents. Figure 11 shows an example. Here, the actions are already numbered with the only logical numbering that is also in line with the requirements. According to Requirement I, the splits x and y can have only the number 0 or 1. On the other hand, according to Requirement III(c), the join v must have the number 4, although its parents have numbers 0 and/or 1. This is why the case when one of the parents of a join is a split is excluded from Requirement III(g).

As can be seen from this discussion, defining a useful and consistent set of requirements for general workflow graphs is not easy. It is not obvious that Requirements I–III can always be fulfilled. However, in the following, we shall prove that Requirements I–III can be fulfilled with the help of a slightly modified version of Algorithm 1 that is shown as Algorithm 2.

Remark 1. The only difference between Algorithm 1 and Algorithm 2 is that, in the former, the command "$n(x) := num$" is within the "**if** $x \in A$" block, whereas in the latter this command is carried out for all vertices. Otherwise the algorithm is the same.

Theorem 3. *For any workflow graph, Algorithm 2 results in an extended numbering that fulfils Requirements I–III.*

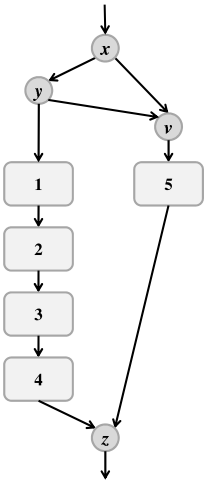


Fig. 11. Example where the number of a join is not determined by the numbers of its parents.

Algorithm 2 Tailored DFS algorithm to compute extended numbering of general workflow graphs

```
procedure DFS(x)
{
    visited(x):=true
    Let xy_1 < ... < xy_k be the edges going out of x
    for(i = k; i > 0; i − −)
        if not visited(y_i)
            DFS(y_i)
    n(x) := num
    if x ∈ A
        num − −
}

procedure main
{
    foreach vertex x
        visited(x):=false
    num:=|A|
    DFS(s)
}
```

Proof. **The result is indeed an extended numbering.** It is clear that each vertex receives a number not greater than $|A|$. Since num is decreased with each action that receives a number, all actions receive different numbers. If the current vertex is not an action, then num remains unchanged, thus the actions receive the numbers $|A|, |A|-1, \ldots, 1$. When the last action has received its number, num reaches 0 and thus every further vertex receives the number 0.

Requirement I. This follows from the well-known properties of DFS (Cormen *et al.*, 2001).

Requirement II. The y_is are not joins, and therefore their only parent is x. Consequently, when the algorithm visits x, none of the y_is can already have been visited. The algorithm first visits y_k. During the call DFS(y_k), the other y_is cannot be visited. Thus, when DFS(y_k) terminates, the next vertex to visit is y_{k-1}. Similarly, after DFS(y_{k-1}) is terminated, y_{k-2} will be visited, and so on. Thus, y_k gets the highest number, y_{k-1} receives a lower number, y_{k-2} gets an even lower number, and so on.

Requirement III(a–f). Since y is not a join, it has exactly one parent, namely x. Hence, the call DFS(y) can only occur from within the call DFS(x). Consequently, after y receives its number $n(y)$ and the call DFS(y) terminates, the algorithm must be within the call DFS(x). Since x is either not a split or, if it is a split, then y is its left-most child, it follows that x has no further children to be visited after y. Thus, x receives right after this its own number. If y is an action, then num was decreased, so in this case $n(x) = n(y) - 1$. Otherwise, i.e., if y is a split, then num was not decreased, so that $n(x) = n(y)$.

Requirement III(g). The call DFS(y) is started from one of the DFS(x_i) calls, say, from DFS(x_{i_0}). Consider the moment when y receives its number $n(y)$. Until this moment, none of the DFS(x_i) calls can have ended. Now the call DFS(y) ends. Since x_{i_0} is not a split, it has no further children to visit, and hence x_{i_0} receives its number immediately. Thus, $n(y) = n(x_{i_0})$. All other x_is will receive their numbers later, and hence their numbers will be not higher than $n(x_{i_0})$, thus $n(y) = n(x_{i_0}) = \max\{n(x_i) : i = 1, 2, \ldots, k\}$. ■

Corollary 2. *In any workflow graph, there is always an extended numbering that fulfills Requirements I–III.*

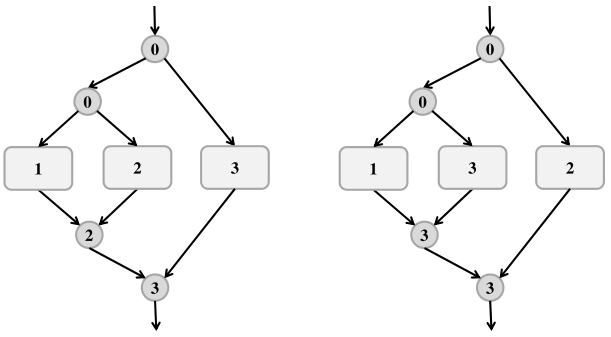


Fig. 12. Two numberings of the same workflow graph.

Similarly to the special case of structured workflow graphs, we managed to prove that a numbering fulfilling the defined set of requirements always exists and can be found efficiently. However, in contrast to Requirements 1–2 for structured workflow graphs, the numbering is not uniquely determined by Requirements I–III, not even for structured workflow graphs. This can be seen in Fig. 12, which shows two numberings of the same structured workflow graph, both fulfilling Requirements I-III.

Finally, as a positive result, we can prove that Requirements I–III are a generalization of Requirements 1–2 in the following sense.

Theorem 4. *Let G be a structured workflow graph and $n : A \to \{1, 2, \ldots, |A|\}$ a numbering that fulfills Requirements 1–2. Then there is an extended numbering $n' : V \to \{0, 1, \ldots, |A|\}$, such that $\forall x \in A : n'(x) = n(x)$ and n' fulfills Requirements I–III.*

Proof. According to Theorems 1 and 2, Algorithm 1 produces the numbering n. Let n' be the numbering produced by Algorithm 2. According to Remark 1, the only difference between the two algorithms is that Algorithm 2 also numbers the non-action nodes. However, the actions receive the same numbers in both cases, and thus $\forall x \in A : n'(x) = n(x)$. Because of Theorem 3, it is also clear that n' fulfills Requirements I–III. ■

5. Case study

We implemented the presented Algorithm 2 as an extension to ARIS. ARIS is a full-featured business process modelling and management tool that uses a central database to store all items of all process models with all their attributes. The rich feature set of ARIS can be extended with custom scripts. For this purpose, ARIS provides a script language based on Visual Basic and a simple programming environment. ARIS also provides an object model, through which the script can access practically everything that a user can access: the workflow graphs, vertices and edges of the graphs, attributes of all objects, etc.

With this machinery in place, it was quite straightforward to implement the algorithm. As usual in DFS implementations, instead of the recursive variant presented in Algorithm 2, we implemented an iterative variant that uses a stack data structure to store the vertices whose processing has started but has not yet finished.

The runtime of the algorithm is obviously linear in the size of the graph. However, the ARIS implementation was quite slow, requiring 2–3 seconds to process a graph with 40–50 vertices. This is due to technical issues:

- Initializing the script interpretation engine seems to incur a significant up-front time penalty.
- The algorithm needs continually access to the vertices and edges of the graph, requiring frequent database access. In our installation, the database was on a remote computer, and thus network latency had an impact on the runtime of the algorithm.

Despite the slowness (that could be mitigated with appropriate measures if needed), the implementation is a useable prototype. Indeed, we used it successfully in a

Table 1. Summary of results.

	Structured workflow graph, Requirements 1–2	General workflow graph, Requirements I–III
Requirements can always be fulfilled	yes	yes
Requirements determine numbering uniquely	yes	no
Appropriate numbering can be computed efficiently	yes	yes

business process consultancy project at a Hungarian bank, in which we created process models (EPCs) of about 100 business processes. Most processes contained 10–20 action vertices, but there were also processes with about 50 action nodes, spanning 5–6 pages.

The experts involved in the project—both external consultants and employees of the bank—found the numbering generated by the algorithm logical and easy to follow. The consultants found it especially useful that, after changes to the process model, they could restore the logical numbering by pressing a button.

6. Conclusions and future work

This paper is a first attempt to define what a logical numbering of the actions in a workflow graph should be like. Two sets of requirements were defined: one for the important special case of structured workflow graphs and one for general workflow graphs. Table 1 summarizes the results.

As a future research direction, it would be desirable to enhance the set of requirements for general workflow graphs. One specific goal could be to propose a modified set of requirements for general workflow graphs that maintains the positive characteristics of Requirements I–III and at the same time determines the numbering uniquely (i.e., to change the 'no' in the last column of Table 1 also to 'yes'). Alternatively, it would also be useful if the requirements for structured workflow graphs could be generalized to general workflow graphs in such a way that, at least for structured workflow graphs, the numbering would remain unique.

Acknowledgment

I am grateful to my colleague László Dénes for drawing my attention to this problem and for the valuable discussions on what can be considered a logical numbering of a workflow graph.

This work was partially supported by the Hungarian National Research Fund and by the National Office for Research and Technology (Grant No. OTKA 67651).

References

Cormen, T.H., Leiserson, C.E., Rivest, R.L. and Stein, C. (2001). *Introduction to Algorithms*, MIT Press, Cambridge, MA.

Kiepuszewski, B., ter Hofstede, A. and van der Aalst, W. (2003). Fundamentals of control flow in workflows, *Acta Informatica* **39**(3): 143–209.

Microsoft Corporation (2010). Microsoft Office Visio: Number the shapes in a flowchart, `http://office.microsoft.com/en-us/visio/HP866500731033.aspx`.

RFF Electronics (2004). RFFlow user's guide, `http://www.rff.com`.

Scheer, A.-W. (2000). *ARIS—Business Process Modeling*, Springer, Berlin.

Scheer, A.-W., Thomas, O. and Adam, O. (2005). Process modeling using event-driven process chains, *in* M. Dumas, W.M.P. van der Aalst and A.H.M. ter Hofstede (Eds.), *Process-Aware Information Systems*, John Wiley & Sons, Hoboken, NJ, pp. 119–145.

van der Aalst, W.M.P. (1999). Formalization and verification of event-driven process chains, *Information and Software Technology* **41**(10): 639–650.

Vanhatalo, J., Völzer, H. and Koehler, J. (2009). The refined process structure tree, *Data & Knowledge Engineering* **68**(9): 793–818.

Weber, I., Hoffmann, J. and Mendling, J. (2008). Beyond soundness: On the semantic consistency of executable process models, *Proceedings of the 6th European Conference on Web Services, Dublin, Ireland*, pp. 102–111.

Zoltán Ádám Mann received the M.Sc. and Ph.D. degrees in computer science from the Budapest University of Technology and Economics in 2001 and 2005, respectively. He also received an M.Sc. degree in mathematics from Eötvös Loránd University in 2004. Currently, he is an associate professor at the Department of Computer Science and Information Theory, Budapest University of Technology and Economics. In addition, he works as an IT and management consultant for various client organizations. His main research interests are combinatorial algorithms and algorithmic complexity.

Received: 10 November 2009
Revised: 24 April 2010