An integer programming based approach for diagnosing workflows

An Integer Programming based Approach for Diagnosing Workflows

Rik Eshuis[1] and Akhil Kumar[2]

[1] Eindhoven University of Technology, School of Industrial Engineering
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
h.eshuis@tue.nl
[2] Department of Supply Chain and Information Systems, Smeal College of Business,
Penn State University, University Park, PA 16802, USA
akhilkumar@psu.edu

Abstract. Workflow analysis is indispensable to capture modeling errors in workflow designs. While in the past several analysis approaches for workflows have been defined, these approaches do not give precise feedback, making it hard for a designer to pinpoint the exact cause of modeling errors. In this paper we introduce a novel approach for analyzing and diagnosing workflows based on integer programming (IP). Each workflow model is translated into a set of IP constraints. Faulty control flow connectors can be easily detected using the approach by relaxing the corresponding constraints. We show that this approach is correct, and illustrate it with realistic examples where the CPLEX tool is used to solve the IP formulations. Moreover, the approach is flexible and can be extended to handle a variety of new constraints, as well as to support new workflow patterns. Its features complement those of existing approaches.

1 Introduction

A critical challenge in workflow modeling lies in the verification of workflow models [1, 3, 4, 6, 7, 10, 14–17]. These workflow models are typically directed graphs, in which some special control flow connector nodes are used to indicate splits and joins, which can specify *parallel* branches (AND split/join) and *exclusive* branches (called OR split/join). As business processes become more complex, verification assumes greater importance. Typically, workflow graphs are unstructured: splits and joins of type AND or OR are used in arbitrary ways, leading to workflow models with goto like constructs and parallelism. Consequently, it is hard to detect errors in workflow designs by merely inspecting the syntax of the workflow.

Several authors have recognized this problem, and starting with Sadiq and Orlowska [16], have defined approaches for analyzing workflow models [1, 3, 4, 6, 7, 10, 14, 15, 17]. The approaches typically detect two types of error [16]. One is that only some branches of an AND join are activated, leading to deadlock at the join node. Second is lack of synchronization at an OR join: multiple

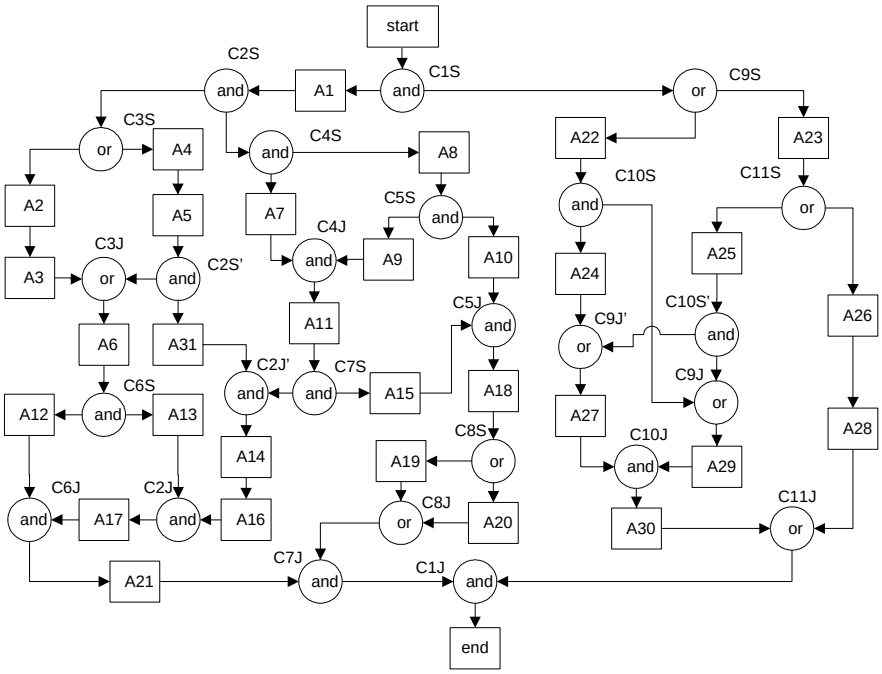

Fig. 1. Example of workflow process from Sadiq and Orlowska [16]

branches of the OR join are activated, leading to multiple instantiations of the join. Other approaches focus on analysis in terms of matched-unmatched pairs and nested-unnested patterns [14], but such analysis presupposes a certain degree of structuredness to be present in workflow models, which is not required by the approach of Sadiq and Orlowska.

Though these approaches help identify the presence of such errors, the feedback they present in case of a found error is rather minimal. To illustrate this point, consider the example in Figure 1, which contains a flaw. Sadiq and Orlowska propose a graph reduction approach to detect such errors. They define a set of rules to reduce a workflow graph; if a single node results, the workflow graph is correct; otherwise, it is incorrect. Applying that approach to this example results in a workflow graph in which all activities are eliminated, and which cannot be reduced any further (see Figure 2). Since the graph is irreducible and consists of more than one node, it is declared incorrect. However, then it is still not clear *which* particular control flow connector is causing the error.

To offer more detailed diagnosis of workflow errors, we present a workflow analysis approach that is based on Integer Programming (IP). Each workflow model is translated into a set of 0/1 linear constraints to which a solution can be found by an IP solver. The approach consists of two consecutive phases:

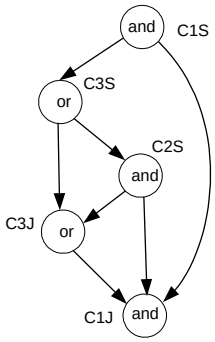

Fig. 2. Diagnosis result of approach [16] for Figure 1

- First, the workflow model is translated into an IP model in which each join is constrained to behave correctly. The IP model is checked for the existence of at least one solution. A solution corresponds to a correct execution. If a solution is found, the workflow model is weakly correct, i.e., it has at least one correct execution.
- Second, the IP formulation is relaxed such that AND and OR joins are allowed to express incorrect behavior, i.e., an AND join can have only some incoming activated branches, and an OR join can have multiple incoming activated branches. The complete workflow model is diagnosed by testing each join in turn, and constraining it to behave incorrectly. If an execution instance exists in which the join behaves incorrectly, it means that this join node is flawed. The workflow designer can use that information to repair the workflow model. If no join behaves incorrectly, the workflow model is strongly correct.

By applying this approach to the example in Figure 1, we find that AND join C2J' and all subsequent AND joins on the path from C2J' to the end node are causing the problem. Section 5 explains the diagnosis for this particular example in more detail. Note that C2J' was eliminated in the approach of Sadiq and Orlowska and is not part of the irreducible graph that their approach yields (cf. Figure 2).

The IP approach is correct, and also very efficient because there are well-known and fast solution procedures for very large IP problems. We also show that our approach is very flexible. In addition to checking for strict/weak correctness and producing detailed diagnosis, it can perform additional user-defined semantic analysis on the workflow as well as deal with workflows described with new routing constructs, beyond AND/OR nodes; see Section 6. Moreover, our approach is complete, so each correct workflow graph can be analyzed using our approach, which is not the case for the original approach of Sadiq and Orlowska [1, 13].

This paper is organized as follows. Section 2 formalizes workflow graphs and their executions. Also the notions of weak and strong correctness are formally defined. Section 3 defines the IP formulation of a workflow graph for checking

weak correctness, and relates it to the formal definitions in Section 2. This defines the first phase of our approach. Next, we illustrate how the CPLEX tool is used to solve the IP models for two running examples. Section 4 defines a relaxation of the IP formulation which is used for diagnosis. Section 5 presents the second phase of the approach by defining a diagnosis algorithm which uses the relaxed IP formulation. We illustrate the diagnosis with the two running examples. Section 6 sketches extensions of our approach. Section 7 presents related work and Section 8 ends with conclusion.

2 Definitions

In this section, we first define a workflow graph for describing a workflow such as the one in Figure 1. We also define an instance subgraph that corresponds to an actual execution instance of this workflow. Then we discuss what it means for a workflow graph to be correct.

2.1 Workflow Graphs

Definition 1. *A workflow graph or schema is a tuple $P = (N, E)$ where:*

- *N is a set of nodes, partitioned into disjoints sets of (X)OR splits S_O, AND splits S_A, (X)OR joins J_O, AND joins J_A, activities (tasks) Act, and $\{start, end\}$ where start is the begin node and end the final node;*
- *$E \subseteq N \times N$ is a precedence relation.*

Auxiliary functions are $inedge, outedge : N \to \mathcal{P}(E)$. Given a node $n \in N$, let $inedge(n) = \{(x, y) \in E \mid y = n\}$ and $outedge(n) = \{(x, y) \in E \mid x = n\}$.

In Figure 1, all nodes are part of N and all edges are in E. Start, end, and activity nodes are depicted by a box, while circles are used to indicate control flow connectors, i.e., splits and joins. The type (AND/OR) is written inside the connector. Note that we only consider exclusive OR.

Next, each workflow graph should satisfy the following structural constraints:

- the start node has no incoming edge and one outgoing edge

$$|inedge(start)| = 0 \wedge |outedge(start)| = 1;$$

- the end node has one incoming edge and no outgoing edge

$$|inedge(end)| = 1 \wedge |outedge(end)| = 0;$$

- each activity node has one incoming and one outgoing edge

$$\forall a \in Act : |inedge(a)| = |outedge(a)| = 1;$$

- each split node has one incoming and two or more outgoing edges

$$\forall s \in S_O \cup S_A : |inedge(s)| = 1 \wedge |outedge(s)| > 1;$$

– each join node has two or more incoming edges and one outgoing edge

$$\forall s \in J_O \cup J_A : |inedge(s)| > 1 \wedge |outedge(s)| = 1;$$

– each node is on a path from the start to the end node (connectedness)

$$\forall n \in N : start E^* n \wedge n E^* end;$$

– there are no loops

$$\forall n \in N : \neg n E^+ n.$$

The first five constraints ease the presentation and can be relaxed without any problems. In the sixth and last constraint, E^* denotes as usual the reflexive-transitive closure of E and E^+ the irreflexive-transitive closure of E. The sixth constraint rules out unconnected workflows because such workflow graphs are already flawed by default, since they contain unreachable parts. The last constraint is also placed by other works on workflow verification [16, 1, 17].

2.2 Instance Subgraphs

An instance subgraph corresponds to an *execution instance* of a workflow graph. An execution instance subgraph is inductively built, starting from the start node. Most rules are self explanatory. If an AND node and all its incoming edges are active, its outgoing edges are made active. If an OR node and one of its incoming edges is active, one of its outgoing edges is made active. If an activity is reached, its outgoing edge is also in the instance subgraph. Note that other approaches [1, 16, 17] only informally define instance subgraphs.

Definition 2. *Let (N, E) be a workflow graph. An instance subgraph is a tuple (N', E') such that:*

IS0 $N' \subseteq N$;
IS1 $E' \subseteq E$;
IS2 $start \in N'$ *and* $outedge(start) \subseteq E'$;
IS3 $n \in N'$ *and* $n \in Act \Rightarrow outedge(n) \subseteq E'$;
IS4 $n \in N'$ *and* $n \in S_A \cup J_A$ *and* $inedge(n) \subseteq E' \Rightarrow outedge(n) \subseteq E'$;
IS5 $n \in N'$ *and* $n \in S_O \cup J_O$ *and* $|inedge(n) \cap E'| = 1 \Rightarrow |outedge(n) \cap E'| = 1$;
IS6 $(x, y) \in E' \Rightarrow x, y \in N'$;
IS7 $n \in N'$ *and* $inedge(n) \cap E' = \emptyset \Rightarrow n = start$.

The last constraint, IS7, is needed in order to allow only valid instance subgraphs. It requires that every node present in the subgraph, other than the start node, must have an incoming edge also in the instance subgraph. Figure 3 (a) shows an example of an invalid instance subgraph (represented by the solid lines), which satisfies IS0-IS6 but not IS7. Figure 3 (b) shows an example of a valid instance subgraph that satisfies all constraints of Definition 2.

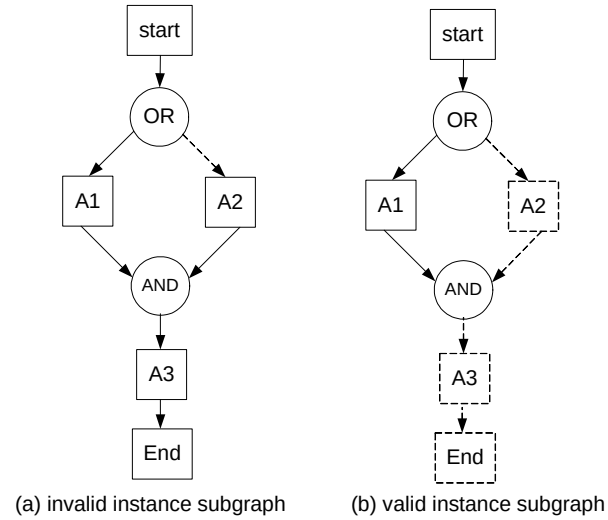

Fig. 3. Example of invalid and valid instance subgraph (solid lines represent subgraphs)

2.3 Correctness

A workflow may be incorrect due to deadlock or a lack of synchronization, as explained in Section 1. Translated to instance subgraphs, this means that an instance subgraph is incorrect if it contains a join whose outgoing edges are not activated, so the left hand side of either IS4 (for an AND join) or IS5 (for an OR join) is not satisfied. An instance subgraph is correct if it has no such joins. Note that an incorrect instance subgraph is valid, since IS4 and IS5 evaluate to true if their left hand side is false.

Definition 3. *An instance subgraph* (N', E') *is correct if and only if for each* $n \in N'$*:*

- *if* $n \in J_A$ *then* $inedge(n) \subseteq E'$*;*
- *if* $n \in J_O$ *then* $|inedge(n) \cap E'| = 1$*.*

The following lemma follows from the definitions.

Lemma 1. *If an instance subgraph* (N', E') *is correct, then* $end \in N'$*.*

Proof. Suppose $end \notin N'$. Since (N, E) is connected, then there is a node $n \in N'$ such that $outedge(n) \cap E' = \emptyset$. By definition of instance subgraph, n is either an AND join or an OR join. However, then n violates the definition of correctness. □

The reverse direction is not true. Figure 4 shows a workflow graph that has a valid yet incorrect instance subgraph that contains node *end*.

Finally, we lift the notion of correctness from instance subgraphs to workflow graphs to distinguish between strong and weak correctness.

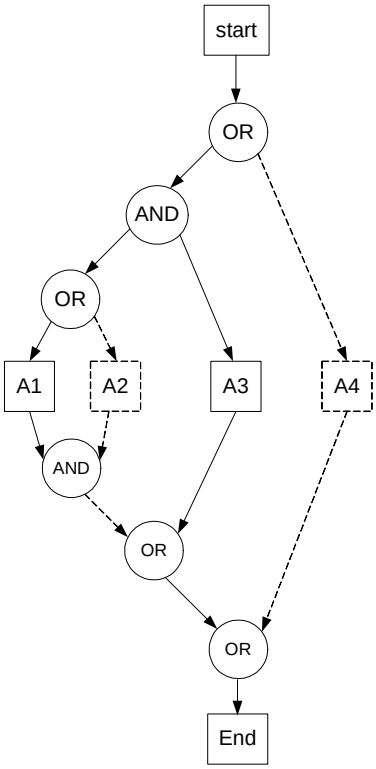

Fig. 4. Example incorrect instance subgraph that contains the end node (instance subgraph shown in solid lines)

Definition 4. *A workflow graph* (N, E) *is* strongly correct *if and only if all its instance subgraphs are correct. A workflow graph is* weakly correct *if and only if at least one of its instance subgraphs is correct.*

The workflow graph in Figure 4 is weakly correct (a correct instance subgraph exists which contains A4) but not strongly correct.

3 Basic IP Model for Weak Correctness

In this section we develop the first phase of our formal approach for verifying workflow graphs using integer programming. We define an IP formulation for workflow graphs. In the formulation, each workflow graph is translated into a set of IP constraints. We show that a solution to this formulation is a correct instance subgraph. Next, we explain how we used CPLEX to test the IP formulation on two non-trivial examples from the literature.

3.1 Basic Integer Programming Formulation

The main features of the IP formulation for a workflow graph (N, E) are discussed first. The IP constraints ensure that each solution corresponds to a correct instance subgraph of (N, E). For writing the constraints, we assume that each split has two outgoing edges, and each join has two incoming edges. However, this can be easily generalized to more incoming and outgoing edges. We use subscripts to identify the different elements of $inedge(n)$ and $outedge(n)$. For example, if $inedge(n) = \{e_1, e_2\}$, then $inedge_1(n) = e_1$ and $inedge_2(n) = e_2$.

Definition 5. *For a workflow graph (N, E), the **Basic IP formulation** maximizes the value at the final node end subject to the following constraints at each node in the workflow graph. Each IP variable (in* `typewriter font`*) corresponds to a node or edge of (N, E):*

IP0 `start = 1`
IP1 For $n \in Act \cup S_A \cup S_O \cup \{end\}$: $\texttt{inedge}_1\texttt{(n) - n = 0}$
IP2 For $n \in Act \cup J_A \cup J_O \cup \{start\}$: $\texttt{outedge}_1\texttt{(n) - n = 0}$
IP3 For $n \in S_A$: $\texttt{outedge}_1\texttt{(n) + outedge}_2\texttt{(n) - 2n = 0}$
IP4 For $n \in J_A$: $\texttt{inedge}_1\texttt{(n) + inedge}_2\texttt{(n)- 2n = 0}$
IP5 For $n \in S_O$: $\texttt{outedge}_1\texttt{(n) + outedge}_2\texttt{(n) - n = 0}$
IP6 For $n \in J_O$: $\texttt{inedge}_1\texttt{(n) + inedge}_2\texttt{(n) - n = 0}$

The rationale behind the rules is as follows. IP0 states that the start node is always part of the solution. The remaining rules consist of symmetrical pairs. IP1 (IP2) ensures that a node with a single incoming (outgoing) edge is active if and only if the incoming (outgoing) edge is active. IP3 (IP4) states that an AND split (join) is active if and only if all its outgoing (incoming) edges are active. IP5 (IP6) states that an OR split (join) is active if and only if one of its outgoing (incoming) edges is active.

A solution to the IP formulation assigns a 0-1 integer value to each node and edge variable in the graph. The nodes and edges that are assigned a 1 value are present in an instance subgraph, and the others are not. We call such a subgraph *an IP subgraph*.

To show the correctness of the IP formulation, we first give an alternative definition (IP-prop) in terms of propositional logic. The IP-prop definition is proven to be equivalent to both the IP formulation as well as the definition of correct instance subgraph.

Definition 6. *Let (N, E) be a workflow graph. The modified formulation (IP-prop) translates the 0/1 constraints in the Basic IP formulation into propositional statements:*

IP-prop0 $start \in N'$
IP-prop1 *For* $n \in Act \cup S_A \cup S_O \cup \{end\}$*:* $n \in N' \Leftrightarrow inedge_1(n) \in E'$
IP-prop2 *For* $n \in Act \cup J_A \cup J_O \cup \{start\}$*:* $n \in N' \Leftrightarrow outedge_1(n) \in E'$
IP-prop3 *For* $n \in S_A$*:* $n \in N' \Leftrightarrow outedge_1(n), outedge_2(n) \in E'$
IP-prop4 *For* $n \in J_A$*:* $n \in N' \Leftrightarrow inedge_1(n), inedge_2(n) \in E'$

IP-prop5 *For* $n \in S_O$*:* $n \in N' \Leftrightarrow outedge_1(n) \in E' \vee outedge_2(n) \in E' \wedge \neg(outedge_1(n) \in E' \wedge outedge_2(n) \in E')$

IP-prop6 *For* $n \in J_O$*:* $n \in N' \Leftrightarrow inedge_1(n) \in E' \vee inedge_2(n) \in E' \wedge \neg(inedge_1(n) \in E' \wedge inedge_2(n) \in E')$

An IP-prop subgraph is a subgraph of the workflow graph that satisfies the IP-prop constraints.

The following lemma asserts that IP subgraphs and IP-prop subgraphs are equivalent. This shows that the IP formulation and the IP-prop formulation are equivalent.

Lemma 2. *A subgraph* (N', E') *is an IP subgraph if and only if* (N', E') *is an IP-prop subgraph.*

Proof. We prove that for each node $n \in N$, `n=1` $\Leftrightarrow n \in N'$, and for each edge $e = (x, y) \in E$, `e=1` $\Leftrightarrow (x, y) \in E'$. We only show the proof for rules IP5 and IP-prop5. The other rules are by similar reasoning.

$\Rightarrow$: Let $n \in S_O$ be an OR split. Suppose $\texttt{outedge}_1\texttt{(n)+outedge}_2\texttt{(n)-n=0}$. There are three cases:

- $\texttt{outedge}_1\texttt{(n)} = 0$, $\texttt{outedge}_2\texttt{(n)} = 0$, $\texttt{n} = 0$. Then $n \notin N'$ and $outedge_1(n)$, $outedge_2(n) \notin E'$. Then IP-prop5 is true.
- $\texttt{outedge}_1\texttt{(n)} = 0$, $\texttt{outedge}_2\texttt{(n)} = 1$, $\texttt{n} = 1$. Then $n \in N'$ and $outedge_2(n) \in E'$ and $outedge_1(n) \notin E'$. Then IP-prop5 is true.
- $\texttt{outedge}_1\texttt{(n)} = 1$, $\texttt{outedge}_2\texttt{(n)} = 0$, $\texttt{n} = 1$. Then $n \in N'$ and $outedge_1(n) \in E'$ and $outedge_2(n) \notin E'$. Then IP-prop5 is true.

$\Leftarrow$: Suppose IP-prop5 is true. There are three cases:

- $n \notin N'$ and $outedge_1(n), outedge_2(n) \notin E'$. Then $\texttt{n} = \texttt{outedge}_1\texttt{(n)} = \texttt{outedge}_2\texttt{(n)} = 0$. So IP5 holds.
- $n \in N'$, $outedge_2(n) \in E'$ and $outedge_1(n) \notin E'$. Then $\texttt{n} = \texttt{outedge}_2\texttt{(n)} = 1$ and $\texttt{outedge}_2\texttt{(n)} = 0$. So IP5 holds.
- $n \in N'$, $outedge_1(n) \in E'$ and $outedge_2(n) \notin E'$. Then $\texttt{n} = \texttt{outedge}_1\texttt{(n)} = 1$ and $\texttt{outedge}_1\texttt{(n)} = 0$. So IP5 holds.

□

The following lemma asserts the equivalence of IP-prop subgraphs and correct instance subgraphs.

Lemma 3. *A subgraph* (N', E') *is a correct instance subgraph if and only if* (N', E') *is an IP-prop subgraph.*

Proof. $\Rightarrow$: Let (N, E') be a correct instance subgraph, so satisfying IS0-IS7. We need to check that IP-prop0 – IP-prop6 holds.

- IP-prop0. Follows from IS2.
- IP-prop1. $\Rightarrow$: Follows from IS7. $\Leftarrow$: Suppose $inedge_1(n) \in E'$. By IS6, $n \in N'$.
- IP-prop2. $\Rightarrow$: Follows from IS3, IS4, IS5. $\Leftarrow$: Follows from IS6.

- IP-prop3. $\Rightarrow$: By IP-prop1, $inedge_1(n) \in E'$ and by the syntactic constraints, $|inedge(n)| = 1$. So the precondition of IS4 is satisfied, therefore $outedge(n) \subseteq E'$. This is equivalent to the righthand side of IP-prop3. $\Leftarrow$: Follows from IS6.
- IP-prop4. $\Rightarrow$: Follows from IS4 and the definition of correctness. $\Leftarrow$: Follows from IS6.
- IP-prop5. $\Rightarrow$: By IP-prop1, $inedge_1(n) \in E'$ and by the syntactic constraints, $|inedge(n)| = 1$. So the precondition of IS5 is satisfied, therefore $|outedge(n) \cap E'| = 1$. $\Leftarrow$: Follows from IS6.
- IP-prop6. $\Rightarrow$: Follows from IS5 and the definition of correctness. $\Leftarrow$: Follows from IS6.

($\Leftarrow$). Sketch: let (N', E') be an IP-prop subgraph, so satisfying IP-prop1 – IP-prop6. We need to check that (N', E') satisfies IS0-IS7 and is correct.

- IS0 and IS1. Follows from Definition 6.
- IS2. Follows from IP-prop0 and IP-prop2.
- IS3. Follows from IP-prop2.
- IS4. Follows from IP-prop2 and IP-prop3.
- IS5. Follows from IP-prop2 and IP-prop5.
- IS6. Follows from IP-prop1 – IP-prop6.
- IS7. Suppose that there is a node $n \in N'$ such that $inedge(n) \cap E' = \emptyset$. From IP-prop1, IP-prop4, and IP-prop6 follows that $n = start$.
- Correctness (Def. 3). Follows from IP-prop4 and IP-prop6.

□

The following lemma shows that an IP subgraph corresponds to a correct instance subgraph, and thus shows the correctness of the Basic IP formulation.

Lemma 4. *An instance subgraph (N', E') is correct if and only if (N', E') is an IP subgraph.*

Proof. Follows immediately from Lemma 2 and 3. □

3.2 Using CPLEX

By solving the IP formulation described above, we can find an instance that represents a correct execution path for the workflow process. We tested our approach on two non-trivial examples from the literature. First, consider the example shown in Figure 1 from [16]. We first modified this figure by removing all activity nodes and assigning an integer number as a label to each node. This results in the graph shown in Figure 5. This graph was transformed into an IP formulation with a short Python program. Finally, the IP was solved using CPLEX [8]. The solution produced by it corresponds to an execution instance, and it is shown in the figure by solid lines. The dashed lines belong to the workflow graph but not to the subgraph.

A second example we ran was taken from [13]. This process is shown in Figure 6 and it was chosen because according to Lin et al. [13] this correct workflow

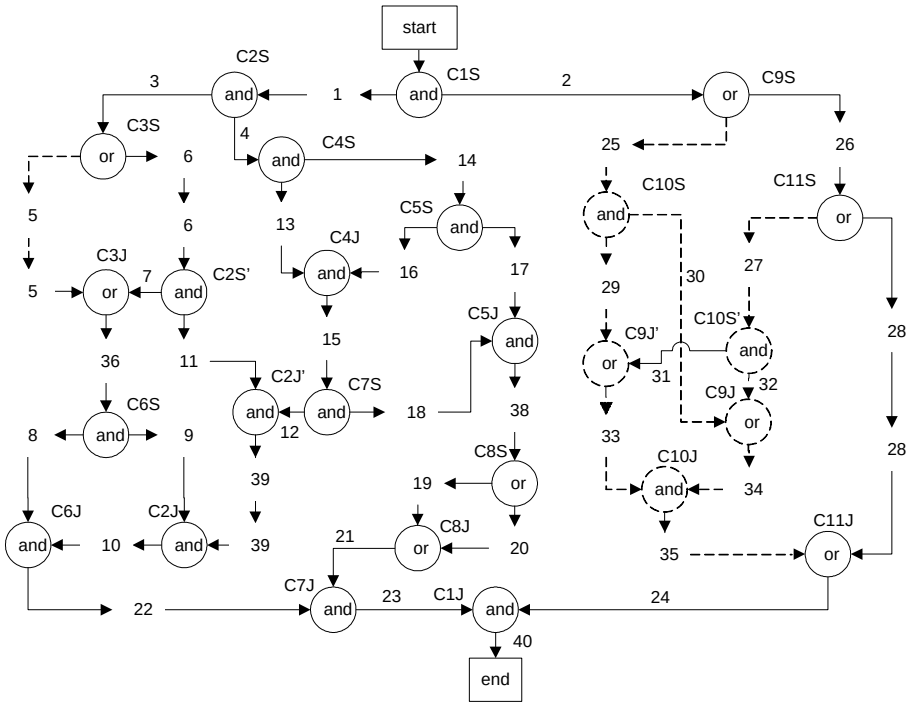

Fig. 5. Modified workflow graph from Figure 1 after removing activity nodes and adding edge labels (correct instance subgraph shown in solid lines)

graph cannot be verified by the approach of [16]. Here activity nodes have been omitted. This workflow graph was also converted into an IP formulation as above. The solution of the IP formulation, i.e., a correct instance subgraph, is shown in the figure by the solid lines.

4 Relaxed IP Model for Diagnosis

Above, we have shown how a workflow graph can be modeled formally using an IP approach. This IP model can be solved with tools like CPLEX [8]. While the above formulation is elegant, it can only tell us if a correct instance subgraph exists for the workflow graph. If not, then CPLEX reports that a solution is infeasible but it does not confirm whether the workflow graph is strongly correct or not. So, if the user wants further diagnosis and verification, she has to go further. Here, we relax the IP formulation slightly to a new formulation (**IPRelax**) in order to test each join for errors. We treat the cases for AND joins and OR joins separately. In the next section, **IPRelax** is used to diagnose workflow graphs and to do strong correctness checking.

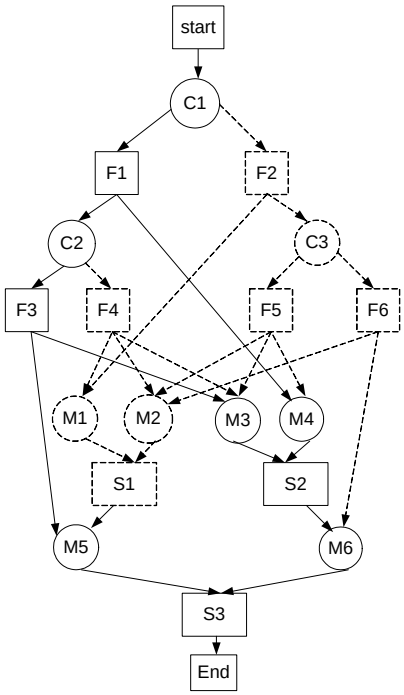

Fig. 6. Example workflow graph from Lin et al. [13] (correct instance subgraph shown in solid lines)

4.1 AND Joins

At every AND join $n \in J_A$, we relax constraint IP4 to:

$$\texttt{inedge}_1\texttt{(n)} + \texttt{inedge}_2\texttt{(n)} - \texttt{n} \leq 1$$

This constraint forces the value of a node to be 1 when both its incoming edges are 1. In addition, we need to add two more constraints at the AND join:

$$\texttt{n} \leq \texttt{inedge}_1\texttt{(n)}$$
$$\texttt{n} \leq \texttt{inedge}_2\texttt{(n)}$$

With these constraints, if both incoming edges are 0, or only one incoming edge is 1, then the node value is forced to be 0. Therefore, for all combinations of values for incoming edges, we can determine a correct value for the node. It is straightforward to extend this to the case of more than two incoming edges.

4.2 OR Joins

In a similar way, we also relax the strict equality constraint at the OR join nodes (IP6) to allow for multiple incoming edges to be active. The revised constraint

is, for $n \in J_O$:

$$\text{(ORJ1)} \qquad \texttt{inedge}_1\texttt{(n) + inedge}_2\texttt{(n) - n} \geq 0$$

This constraint prevents an OR join node from being 1 unless at least one incoming edge is 1. In addition three consistency constraints are added:

$$\begin{aligned} &\text{(ORJ2)} \qquad \texttt{n} \geq \texttt{inedge}_1\texttt{(n)} \\ &\text{(ORJ3)} \qquad \texttt{n} \geq \texttt{inedge}_2\texttt{(n)} \\ &\text{(ORJ4)} \qquad \texttt{inedge}_1\texttt{(n) + inedge}_2\texttt{(n)+ n} \leq 2 \end{aligned}$$

(ORJ2) and (ORJ3) force node n to be active if at least one of the incoming edges is active. However, (ORJ4) requires that only and exactly one incoming edge of node n is active. If more than one incoming edge is active, then the value of the OR join node becomes 0 to reflect that there is a problem at the OR join node. Again, the extension in case n has more than two incoming edges is straightforward.

5 Diagnosis Algorithm and Results

In this section, we discuss the second phase of our approach, in which we do detailed diagnosis of workflow graphs as well as checking of strong correctness. In Section 5.1, we describe our diagnosis algorithm, Section 5.2 gives the results from running the algorithm, and Section 5.3 provides a discussion, in which we highlight a few specific features of the IP approach by contrasting it to [1, 16].

5.1 Algorithm Description

In Section 4, we showed how one can check if a process is strongly correct, i.e. there is no instance subgraph that corresponds to an erroneous execution instance of the process. If there is even one such execution instance then it is necessary to identify it and notify the user. The algorithm in Figure 7 performs this procedure.

This algorithm first creates an IPRelaxed formulation for the workflow graph (l.2) and the error flag is initialized to false (l.3). Next, the algorithm checks each join node in turn (l.6 and l.14). For each join, it adds a constraint to the IPRelax formulation to force an execution instance that would generate an error at this join (l.4 and l.13). For an AND join, an erroneous situation corresponds to only one incoming edge being activated, resulting in deadlock. The constraint $\texttt{inedge}_1\texttt{(n)+inedge}_2\texttt{(n)=1}$ expresses this (l.5). For an OR join, an error corresponds to multiple incoming edges being activated, resulting in multiple instances. Constraint $\texttt{inedge}_1\texttt{(n)+inedge}_2\texttt{(n)} \geq \texttt{2}$ at line 14 reflects this. After adding the appropriate constraint, we solve the modified IPRelax formulation (l.8 and l.16). If a feasible solution is not found, it means that it is not possible to create any execution instance in which an error will occur at

```
 1: procedure DIAGNOSIS((N, E))
 2:     IP1= make IPRelax formulation for (N, E)
 3:     error = false
 4:     for n ∈ J_A do
 5:         IP2=IP1 plus constraint inedge1(n)+inedge2(n)=1
 6:         sol=solve IP2
 7:         if sol is not null then
 8:             Print"error at node " + n
 9:             Print sol
10:             error = true
11:         end if
12:     end for
13:     for n ∈ J_O do
14:         IP2=IP1 plus constraint inedge1(n)+inedge2(n)≥ 2
15:         sol=solve IP2
16:         if sol is not null then
17:             Print "error at node " + n
18:             Print sol
19:             error = true
20:         end if
21:     end for
22:     if error is false then
23:         Print "The workflow graph is strongly correct"
24:     end if
25: end procedure
```

Fig. 7. Detailed diagnosis algorithm

this node. On the other hand, if a solution is found, it is reported as part of the diagnosis, and the error flag is raised (l.8-10 and l.17-19). The found solution represents an actual instance subgraph that leads to an error arising at this node. This information tells a user exactly where an error can occur and the exact execution path that leads to the error. Finally, if the error flag has not been raised (l.22), so no error at a join was found, the workflow graph is strongly correct.

The lemma below asserts the correctness of the algorithm for the strong correctness check.

Lemma 5. *Let (N, E) be a workflow graph. The algorithm finds no error if and only if (N, E) is strongly correct.*

Proof. $\Rightarrow$: Since algorithm Diagnosis finds no error, for each join there is no instance subgraph in which the join behaves incorrectly, so the conditions in l.7 and l.16 never become true. Thus, (N, E) is correct by Definition 2.

$\Leftarrow$: Suppose the algorithm finds a join j such that IP_2 produces a feasible solution (l.7 and l.15). We only consider the case that j is an AND join, the case that j is an OR join is by similar reasoning. The found solution satisfies $inedge_1(j)+inedge_2(j) = 1$ (l.5), so not all incoming branches of j are activated.

But then the corresponding instance subgraph is not correct, so then (N, E) is not correct. □

Thus, algorithm Diagnosis implements both a detailed diagnosis procedure as well as a strong correctness check. Next, we discuss the results of applying this algorithm.

5.2 Results

We ran algorithm Diagnosis for the two examples discussed above. For the first example (Figure 5), the results are shown in Table 1. These results show that there are five AND joins (C2J', C2J, C6J, C7J, C1J) where a deadlock can occur because only one incoming edge is activated. All these deadlocks however occur because of the same problem which is the AND join node C2J'. The subsequent join nodes lie on a path from C2J' to end. By inspecting the found solution for C2J', we can trace this deadlock back to node C2S', which is not activated if the left branch is taken at OR split node C3S. Hence, the problem can be isolated to the (right) branch from C3S to C2S'. At the OR joins, no problems were found.

Table 1. Diagnosis results on Figure 5 [16]

node	type	constraint	result
C3J	or-join	d5+d7=2	infeasible
C2J'	and-join	d11+d12=1	solution found, d11=0, d12=1,obj=0
C6J	and-join	d8+d10=1	solution found, d8=1, d10=0, obj=0
C4J	and-join	d13+d16=1	infeasible
C5J	and-join	d17+d18=1	infeasible
C2J	and-join	d9+d39=1	solution found, d9=1, d39=0, obj=0
C7J	and-join	d21+d22=1	solution found, d21=1, d22=0, obj=0
C1J	and-join	d23+d24=1	solution found, d23=0, d24=1, obj=0
C8J	or-join	d19+d20=2	infeasible
C9J	or-join	d30+d32=2	infeasible
C9J'	or-join	d29+d31=2	infeasible
C10J	and-join	d33+d34=1	infeasible
C11J	or-join	d28+d35=2	infeasible

We did similar testing on the second example (Figure 6). Here the diagnosis results showed that every join node is correct. Hence, the entire process is correct.

5.3 Discussion

We defined two IP formulations, an exact one to check for existence of a correct instance subgraph, and a relaxed one to check strong correct correctness and do detailed diagnosis. Solving the first formulation corresponds to checking relaxed soundness [5]. The second formulation lets us check for (normal) soundness [1] because it reports any nodes where conditions for soundness may be violated.

Next, we will highlight specific features of the IP approach. Using the example in Figure 1, we compare the IP approach with the reduction algorithm of [16] and the Woflan-based approach of [1]. We will also discuss the performance of the IP approach.

Figure 2 gives the solution found by the reduction algorithm [16] for Figure 1, already discussed in Section 1. The algorithm stops here because none of the reduction rules can be applied anymore. But with our approach, we can diagnose specific joins. Using the representation in Figure 5, join C2J' is diagnosed by adding the constraint: d11 + d12 =1 to the IPRelax formulation. In the resulting instance subgraph (due to space limitations not shown here), the AND join C2J' deadlocks since the right path from C2S' and the left path from C3S are taken. In addition, there are four other AND joins C2J, C6J, C7J and C1J which are deadlocked as a result, because the edges denoted by 39, 10, 22 and 23 (respectively) in Figure 5 all have values of 0 in our solution and fail to get activated. This clearly suggests that the main problem lies at node C2J' and creates a cascading effect resulting in the subsequent problems. The process designer must fix this problem first and then check the process again. This additional diagnosis information is more useful to a user in resolving the problem in the process than the reduced process in Figure 2. However, the reduction rules of Sadiq and Orlowska can be integrated with our approach: first the reduction rules can be applied to the workflow graph, and next the reduced workflow graph can be converted into an IP model. This way, a reduced IP model is obtained. In that sense, our approach complements the approach of Sadiq and Orlowska [16].

Based on Lin [13], Van der Aalst et al. [1] point out that the approach of Sadiq and Orlowska is not complete, and propose to use Petri net-based analysis techniques to diagnose workflows. These techniques have been implemented in the Woflan tool [18], which offers several diagnostics in case of errors. However, these diagnostics are sometimes contradicting and do not always point to the actual error, as we show next. The three main types of Woflan diagnostics are mismatches, locking scenarios, and coverability of threads of control. A mismatch occurs if there is a pair of split and join nodes, connected by two directed disjoint paths, that have different types. Such pairs can be detected from the syntax of the workflow model. Van der Aalst et al. [1] suggest that a deadlock corresponds to an OR split that has a corresponding AND join while a lack of synchronization corresponds to an AND split that has a corresponding OR join. However, applying Woflan to the example of Figure 1 yields 3 mismatched pairs: (C3S,C6J), (C3S,C2J), and (C9S,C10J). None of these pairs identify that C2J' is the main cause of the problem. Moreover, pair (C9S,C10J) belongs to a part of the workflow which is correct. Another diagnostic is a locking scenario, which is series of activities that lead to a point of execution from which no proper termination is possible. For Figure 1, Woflan returns 126 of these scenario, all of which stop after activity A2 has been done. Again, this does not indicate that C2J' is causing the problem. Finally, Woflan computes whether each element of the workflow can be covered by a thread of control, i.e. a sequential state machine. Woflan points out that the left incoming edge of C2J' is not covered

by any thread, but it is not clear how the workflow can be adjusted to repair this. The IP diagnosis also identifies C2J' as cause of the error, but in addition provides an incorrect instance subgraph that illustrates the flaw at C2J'. In sum, for the example in Figure 1, the three Woflan diagnostics provide different, contradicting clues which are not very accurate. In contrast, the detailed diagnosis provided by the IP approach gives exact feedback to the workflow designer.

Another feature of our approach is that the complexity only grows in the number of join nodes. Basically the diagnosis procedure requires that we solve the IP formulation as many times as the number of join nodes. Thus, for the example of [16] there are 13 join nodes in Figure 1, and as shown in Table 1, we solved the IP formulation 13 times and reported the results in order to produce a complete diagnosis. Although theoretically, the IP has exponential complexity, in practice the algorithm runs very fast for two reasons. First, most IP software algorithms first solve the relaxed linear programming version of the problem. If it is infeasible, then there is no need to solve the IP formulation. Secondly, if the relaxed formulation gives an integer solution then it is not necessary to run a more expensive algorithm like branch and bound in order to get the IP solution.

6 Extensions

In this section we discuss two key extensions of our approach. Both of them serve to highlight how this approach provides more flexibility than the other ones such as [16]. The first extension shows how it is possible to perform semantic checking of the workflow graph. The second extension shows how to apply our approach to workflow graphs that contain additional control patterns. Other approaches (notably [16, 18, 1]) we have discussed above do not lend themselves so easily to such extensions.

6.1 Semantic Checking and Analysis

A workflow model may be structurally correct, but it may violate certain basic (business) rules regarding relationships between activities. This means it is semantically incorrect. For example, in an insurance claim process, an application must be received before the claim can be reviewed. Therefore, the activity "application received" must always occur in every instance. Another simple rule is that a client's application cannot both be *accepted* and *denied* for the same process instance. Thus, the two activities "application accepted" and "application rejected" are exclusive. Similarly, another rule is that if an application is received, then one of two activities "application accepted" or "application rejected" must occur. We show here how a variety of such rules can be checked easily using our approach. Such constraints fall into generic categories [9]:

- (C1) Occurrence/Non-occurrence of an activity: Does an activity always (never) appear in at least one execution instance?
- (C2) Co-occurrence of activities in an activity group: Do two activities always (never) appear together or not at all in all execution instances?

– (C3) Dependency relationships between activities: does one task always depend upon another?

The general idea is that to check for each rule or constraint we can add additional constraints to our IP formulation and solve it. Thus, a C1 type constraint simply checks if an activity a (never) appears in an execution instance of a process. To perform this check, we can add a constraint `a = 1` or `a = 0` to the formulation IPrelax and solve it. If a solution is (not) found the answer is true. Instead of the activity we can use the incoming or outgoing edge of the activity because their IP variables will have the same value by the IP constraints. A type C2 constraint checks if a group of n activities $\{a_1, a_2, .., a_n\} \subseteq Act$ always occur together. We can add an IP constraint to the formulation IPrelax as follows:

$$\mathtt{a}_1 \texttt{ + } \mathtt{a}_2 \texttt{ + ... + } \mathtt{a}_n \texttt{= } n$$

If no solution is found then the constraint is true; else, it is false. In a similar way it is possible to check for other kinds of relationships by adding simple IP constraints. Thus, one can analyze the workflow process in considerable detail, and get a better understanding of its behavior. Moreover, we can also show counterexamples where the desired conditions are violated, and this can help to make corrections to the workflow process.

6.2 Adding New Modeling Constructs

The IP formulations in Section 3 and 4 assume that workflows are designed using standard AND and OR patterns. However, the IP formulations can be extended to verification of workflows with new control flow patterns [2]. We illustrate our idea with just two patterns. Consider the discriminator (or multiple instances) pattern which matches an AND split with a discriminator join. In this pattern, multiple outgoing branches are activated at the AND split, and the first branch that finishes can activate the discriminator join. The subsequent branches are simply ignored by the discriminator join when they finish. This ignoring of incoming branches distinguishes the behavior of the discriminator join from the normal exclusive OR join that we use in the definitions of workflow graphs and the IP formulations. The discriminator pattern can be modeled by these constraints (for a discriminator join n with two incoming edges):

$$\mathtt{inedge}_1\mathtt{(n) + inedge}_2\mathtt{(n) - n} \geq \mathtt{0}$$
$$\mathtt{n} \geq \mathtt{inedge}_1\mathtt{(n)}$$
$$\mathtt{n} \geq \mathtt{inedge}_2\mathtt{(n)}$$

These constraints are identical to the relaxed constraints for OR joins (ORJ1, ORJ2, ORJ3) as specified in Section 4.2. However, constraint ORJ4 is not used, since the discriminator join allows the join to be activated if more than one incoming branch is activated, while ORJ4 forbids this.

Next, consider the m-of-n AND split pattern in which a split s activates only m out of its n outgoing branches, where $m \leq n$. Similarly, the m-of-n AND join

pattern specifies that a join j is activated by exactly m out of its n incoming branches. These patterns can be modeled with constraints like:

$$\texttt{outedge}_1\texttt{(s)} + \texttt{outedge}_2\texttt{(s)} + \ldots + \texttt{outedge}_n\texttt{(s)} = m\texttt{*s}$$
$$\texttt{inedge}_1\texttt{(j)} + \texttt{inedge}_2\texttt{(j)} + \ldots + \texttt{inedge}_n\texttt{(j)} = m\texttt{*j}$$

Note that these constraints generalize IP3 and IP4, respectively, of Definition 5.

This shows that it is possible to create IP formulations to check correctness of workflow graphs that use additional patterns. In a similar manner various other control flow patterns can also be added. These flexible ways of modeling may often produce workflows that are not strongly correct but they would conform to weaker notions of correctness.

7 Related Work

In Section 5.3, we already gave a detailed comparison of our approach with [1, 16]. Several alternative approaches for verification of workflow models are discussed in [3, 4, 6, 10, 11, 17]. The verification approaches by Lin et al. [13] and Touré et al. [17] extend the approach of Sadiq and Orlowska. Consequently, like [16] these approaches can detect structural conflicts, but do not give details on the causes of found conflicts. A verification approach based on workflow decomposition is given in [4]. This approach may not be able to verify unstructured workflows that are not decomposable. Kiepuszewski et al. [12] address the possibility that an unstructured workflow can be mapped to a structured one through equivalence preserving transformations, but the discussion is mainly through examples, and lacks generality. Logic-based approaches for workflow verification are discussed by Bi and Zhao [3]. While a propositional logic program can also be represented as an integer program, the latter offers greater ease of representation for verification. For example, a constraint to allow only one (out of n) active incoming edges at an OR join can be written as one IP constraint while it would require n logic constraints. While the logic formulation does constraint satisfaction, the IP can also optimize an objective. Finally, there are approaches for analyzing workflow designs that use model checking [6, 10], but there only one error trace (corresponding to one flawed instance subgraph) is returned, so the feedback is less detailed than in the IP approach (cf. the algorithm in Figure 6).

8 Conclusions

Recognizing the importance and continuing need for effective and efficient techniques for analysis of workflows, we formally developed and tested an IP based approach for verification of workflows represented as workflow graphs. The approach converts a workflow graph into an IP formulation, and solves it using a standard tool. While other approaches for verification already exist, we showed that the approach has some unique features that complement the other approaches very well. In particular, the approach gives precise diagnostic information that helps a workflow designer to correct flaws in workflow models. It can

also be easily adapted for checking arbitrary semantic constraints and for verifying new workflow patterns by adding new constraints to capture their behavior.

Future work includes integrating the developed tool with a graphical workflow design environment to allow the feedback to be more easily interpreted by users. We also envision that a user will be able to specify arbitrary constraints and interactively check if they are true through our tool. We would also like to extend the tool with the ability to verify advanced workflow patterns, as well as give users the ability to add their own patterns in a convenient manner.

References

1. W.M.P. van der Aalst, A. Hirnschall, and H. M. W. Verbeek. An alternative way to analyze workflow graphs. In A. Banks-Pidduck, J. Mylopoulos, C. C. Woo, and M. T. Ozsu, editors, *Proc. of the 14th Int. Conf. on Advanced Information Systems Engineering (CAiSE'02)*, volume 2348 of *Lecture Notes in Computer Science*, pages 535–552. Springer-Verlag, Berlin, 2002.
2. W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.
3. H. Bi and J. L. Zhao. Process logic for verifying the correctness of business process models. In *Proc. of the Int. Conf. on Information Systems (ICIS 2004)*, pages 91–100. Association for Information Systems, 2004.
4. Y. Choi and J. Zhao. Decomposition-based verification of cyclic workflows. In D. A. Peled and Y-K. Tsay, editors, *Proc. of Automated Technology for Verification and Analysis (ATVA 2005)*, volume 3707 of *Lecture Notes in Computer Science*, pages 84–98, Taipei, Taiwan, 2005. Springer-Verlag.
5. J. Dehnert and P. Rittgen. Relaxed soundness of business processes. In K.R. Dittrich, A. Geppert, and M.C. Norrie, editors, *Proc. of the 13th Int. Conference on Advanced Information Systems Engineering (CAiSE'01)*, volume 2068 of *Lecture Notes in Computer Science*, pages 157–170. Springer-Verlag, Berlin, 2001.
6. R. Eshuis and R. Wieringa. Verification support for workflow design with UML activity graphs. In *Proc. Int. Conf. on Software Engineering (ICSE 2002)*, pages 166–176. ACM Press, 2002.
7. A.H.M. ter Hofstede, M.E. Orlowska, and J. Rajapakse. Verification problems in conceptual workflow specifications. *Data and Knowledge Engineering*, 24(3):239–256, 1998.
8. ILOG. ILOG CPLEX 11.010, 2008.
9. W. Janssen, R. Mateescu, S. Mauw, P. Fennema, and P. van der Stappen. Model checking for managers. In D. Dams, R. Gerth, S. Leue, and M. Massink, editors, *Proc. 5th and 6th Int. SPIN Workshops*, volume 1680 of *Lecture Notes in Computer Science*, pages 92–107. Springer, 1999.
10. C.T. Karamanolis, D. Giannakopoulou, J. Magee, and S.M. Wheater. Model checking of workflow schemas. In *Proc. EDOC 2000*, pages 170–181. IEEE Computer Society, 2000.
11. B. Kiepuszewski. *Expressiveness and Suitability of Languages for Control Flow Modelling in Workflows*. PhD thesis, Queensland University of Technology, 2002.
12. B. Kiepuszewski, A.H.M. ter Hofstede, and C. Bussler. On structured workflow modelling. In B. Wangler and L. Bergman, editors, *Proc. CAiSE '00*, volume 1789 of *Lecture Notes in Computer Science*, pages 431–445. Springer, 2000.

13. H. Lin, Z. Zhao, H. Li, and Z. Chen. A novel graph reduction algorithm to identify structural conflicts. In *Proc. of the 35th Ann. Hawaii International Conference on System Science (HICSS-35)*. IEEE Computer Society Press, 2002.
14. R. Liu and A. Kumar. An analysis and taxonomy of unstructured workflows. In W.M.P. van der Aalst, B. Benatallah, F. Casati, and F. Curbera, editors, *Proc. 3rd Conference on Business Process Management (BPM 2005)*, Lecture Notes in Computer Science 3649, pages 268–284, 2005.
15. F. Puhlmann and M. Weske. Investigations on soundness regarding lazy activities. In S. Dustdar, J. L. Faideiro, and A. Sheth, editors, *Proc. Int. Conf. on Business Process Management (BPM 2006)*, volume 4102 of *Lecture Notes in Computer Science*, pages 145–160. Springer-Verlag, Berlin, 2006.
16. W. Sadiq and M.E. Orlowska. Analyzing process models using graph reduction techniques. *Information Systems*, 25(2):117–134, 2000.
17. F. Touré, K. Baïna, and K. Benali. An efficient algorithm for workflow graph structural verification. In *Proc. Int. Conf. on Cooperative Information Systems (CoopIS 2008)*, volume 5331 of *Lecture Notes in Computer Science*, pages 392–408. Springer-Verlag, Berlin, 2008.
18. H.M.W. Verbeek, T. Basten, and W.M.P. van der Aalst. Diagnosing workflow processes using Woflan. *The Computer Journal*, 44(4):246–279, 2001.