Case Completion of Workflows for Process-Oriented Case-Based Reasoning

Gilbert Müller and Ralph Bergmann

Business Information Systems II
University of Trier
54286 Trier, Germany
[muellerg][bergmann]@uni-trier.de,
http://www.wi2.uni-trier.de

Abstract. Cases available in real world domains are often incomplete and sometimes lack important information. Using incomplete cases in a CBR system can be harmful, as the lack of information can result in inappropriate similarity computations or incompletely generated adaptation knowledge. Case completion aims to overcome this issue by inferring missing information. This paper presents a novel approach to case completion for process-oriented case-based reasoning (POCBR). In particular, we address the completion of workflow cases by adding missing or incomplete dataflow information. Therefore, we combine automatically learned domain specific completion operators with generic domain-independent default rules. The empirical evaluation demonstrates that the presented completion approach is capable of deriving complete workflows with high quality and a high degree of completeness.

Keywords: process-oriented case-based reasoning, workflows, workflow completion, case completion, completion operators, completion rules

1 Introduction

In Case-based Reasoning (CBR), knowledge is distributed over four knowledge containers [19], i.e., the case base, the vocabulary, the similarity measure, and the adaptation knowledge. If any of these knowledge containers is not sufficiently specified, this incomplete container can be a burden for the entire CBR application unless another container is able to cover this knowledge gap. Consequently, either the CBR system will not work or propose solutions with major mistakes. Thus, lessening the containers' knowledge gap has been extensively investigated, e.g., by learning of similarity measures [24] or adaptation knowledge [11,7,2,27].

It has been shown that incomplete cases, i.e., cases with missing information, also lead to such a critical knowledge gap [5]. A case base of incomplete cases naturally results in a retrieval of incomplete cases that do not provide all required information. Furthermore, similarity assessment is hampered because necessary attributes are missing. Moreover, incomplete cases may also affect the adaptation knowledge container, if adaptation knowledge is learned automatically from the case base. In this case the adaptation knowledge is also incomplete. Consequently, adaptation capabilities are affected

because adaptation quality is reduced or adaptation is prevented. Hence, incomplete cases may have a critical impact on the entire utilisation of the CBR application. Thus, various approaches have been presented that address case completion [6,1,10,20].

In this paper, we investigate this issue within process-oriented CBR (POCBR) [14], which deals with CBR applications for process-oriented information systems. POCBR aims at assisting domain experts in their work with workflows, in particular by supporting workflow reuse. Two important problems of workflow reuse are the retrieval of similar workflows from potentially large repositories [4] as well as the adaptation of workflows [15]. Recently, work has been presented for POCBR, where adaptation knowledge is also learned automatically from the case base [15,13]. The issue of incomplete cases also exists in the field of POCBR. Our investigations showed that workflows in existing workflow repositories are mostly incomplete. Whilst the control-flow of workflows is mostly fully specified, the dataflow usually is not completely defined or not existent at all. The IWi process model corpus [25], for example, contains more than 4000 process models resulting from various domains and sources. However, in most of these process models the dataflow is not complete. Consequently, employing such incomplete workflow cases may significantly affect retrieval and adaptation capabilities. This problem also arises in textual process descriptions. In the cooking domain, for example, textual descriptions of recipes contain not a full description of the dataflow, i.e., the output of tasks is usually missing. Thus, this issue has also recently been addressed by workflow extraction from textual process descriptions [21].

This paper presents a new approach for completing the dataflow of workflows automatically. Figure 1 summarizes the idea of completing workflow cases to lessen the knowledge gap in the case base as well as in the adaptation knowledge. Prior to inserting workflows into the case base, automatic case completion is applied. Case completion makes use of domain independent default completion rules as well as domain specific completion operators, automatically learned from the case base. Subsequently, adaptation knowledge is learned based on complete cases and thus retrieval as well as adaptation can be based on complete cases resulting in more complete adaptation knowledge.

In order to automatically learn completion operators, an initial case base has to be established that consists of workflows containing substantially defined dataflows. In application scenarios in which available workflows significantly lack the dataflow information, a small set of fully specified workflows may need to be completed manually to be used as training data. The learned completion operators can then be used to automatically complete the remaining workflow cases.

In the next section, our previous work is summarized providing the fundamentals for the presented approach. Then, workflow completion operators are introduced (Sec. 3) and Section 4 describes how they can be learned from the case base automatically. Then, our novel approach to complete workflow cases based on the learned completion operators is presented (see Sec. 5). Finally, we illustrate the feasibility of our approach by an evaluation in Section 6 and sum up the paper with conclusions and future work.

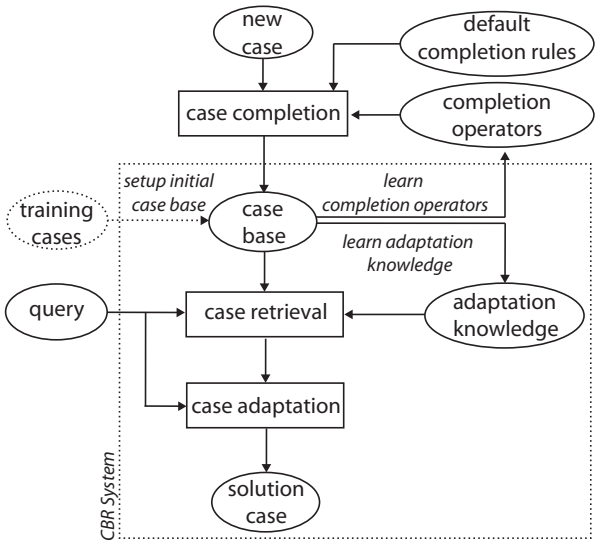

Fig. 1. Framework of case completion adopted from [5]

2 Foundations

We now briefly introduce relevant previous work in the field of POCBR.

2.1 Workflows

Broadly speaking, a *workflow* consists of a set of *activities* (also called *tasks*) combined with *control-flow structures* like sequences, parallel (AND) or alternative (XOR) branches, as well as repeated execution (LOOP). In addition, tasks exchange certain *data items*, which can also be of physical matter, depending on the workflow domain (e.g., ingredients in the cooking domain). Tasks, data items, and relationships between the two of them form the *dataflow*.

We illustrate our approach in the domain of cooking recipes (see example workflow in Fig. 2). A cooking recipe is represented as a workflow describing the instructions for cooking a particular dish [23]. Here, the tasks represent the cooking steps and the data items refer to the ingredients being processed by the cooking steps. An example cooking workflow for a sandwich recipe is illustrated in Figure 2. Based on our previous work [15,17,18] we now introduce the relevant formal workflow terminology.

Definition 1. *A* workflow *is a directed graph $W = (N, E)$ where N is a set of nodes and $E \subseteq N \times N$ is a set of edges. The nodes can either be data nodes N^D, task nodes N^T, or control-flow nodes N^C, i.e., $N = N^D \cup N^T \cup N^C$. In addition, we call $N^S = N^T \cup N^C$ the set of sequence nodes. Edges can be either control-flow edges $E^C \subseteq N^S \times N^S$, which define the order of the sequence nodes or dataflow edges $E^D \subseteq (N^D \times N^S) \cup (N^S \times N^D)$, which define how the data is shared between the tasks, i.e., $E = E^C \cup E^D$.*

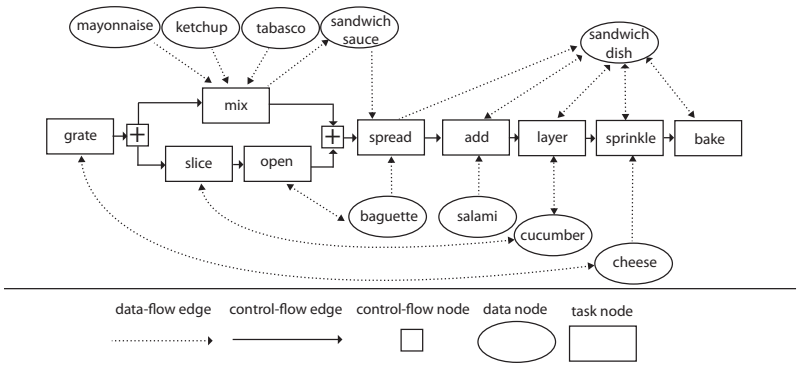

Fig. 2. Example of a block-oriented cooking workflow

Furthermore, nodes have a semantic description from a semantic meta data language Σ, which is assigned by the function $S : N \rightarrow \Sigma$, thus leading to a semantic workflow [4].

The control-flow edges E^C of a workflow induce a strict partial order on the sequence nodes N^S. Thus, we define $s_1 < s_2$ for two sequence nodes $s_1, s_2 \in N^S$ as a transitive relation that expresses that s_1 is executed prior to s_2 in W.

We further denote t^D as the set of data nodes produced by a task t, i.e., $t^D = \{t \in N^T | \exists (t,d) \in E^D\}$, and t_D as the set of data nodes consumed by the task t, i.e., $t_D = \{t \in N^T | \exists (d,t) \in E^D\}$. Data nodes that are produced by a task, but not consumed by this task are referred to as creator data nodes $N^{D*} = \{d \in N^D | \exists t \in N^T : d \in t^D \wedge d \notin t_D\}$. Moreover, data nodes $d_1, d_2 \in N^D$ are dataflow connected $d_1 \ltimes d_2$ if there exists a task that consumes data node d_1 and produces data node d_2. Moreover, $d_1 \ltimes^+ d_2$ denotes that $d_1, d_2 \in N^D$ are transitively dataflow connected:

$$d_1 \ltimes d_2, \text{iff } \exists t \in N^T : (d_1 \in t_D \wedge d_2 \in t^D) \tag{1}$$

$$d_1 \ltimes^+ d_2, \text{iff } d_1 \ltimes d_2 \vee \exists d \in N^D : (d_1 \ltimes d \wedge d \ltimes^+ d_2) \tag{2}$$

2.2 Block-Oriented Workflows

We now restrict the workflow representation to block-oriented workflows [15], i.e., workflows in which the control-flow structures form blocks of nested workflows with an opening and closing control-flow element. These blocks must not be interleaved.

Definition 2. *A* block-oriented workflow *is a workflow in which the control-flow nodes $N^C = N^{C_*} \cup N^{C^*}$ define the control-flow blocks. Each control-flow block has an opening node from N^{C_*} and a related closing node from N^{C^*} specifying either an AND, XOR, or LOOP block. These control-flow blocks may be nested but must not be interleaved and must not be empty.*

Figure 2 shows an example block-oriented workflow, containing a control-flow block with an opening AND control-flow node and a related closing AND control-flow node (denoted by [+]).

As we aim at improving the completeness of incomplete workflows, we introduce a terminology of *consistent block-oriented workflows* to define our perception of complete workflows.[1] According to Davenport, "[...] a process is simply a structured, measured set of activities designed to produce a specific output [...]" [8]. In the following, these specific workflow outputs are denoted as $W^O \subseteq N^D$. In the cooking domain, the specific output is the particular dish produced, i.e., "sandwich dish" in Fig. 2. Hence, for a *consistent workflow*, we require that each ingredient must be contained in the specific output, as otherwise the ingredient as well as the related tasks would be superfluous. Further, identical AND/OR branches as well as several creator tasks producing the same data output usually are not very plausible and must consequently not exist in a consistent workflow. Thus, we define consistent workflows for this paper as follows.

Definition 3. *A block-oriented workflow is* consistent*, if and only if the following conditions hold. Each produced data node is contained in the specific output of the workflow. Thus, each data node must be transitively dataflow connected to the specific output W^O, i.e., $\forall d \in N^D \exists o \in W^O : d \ltimes^+ o$ and further each task has at least one input and one output data object, i.e., $t_D \neq \emptyset$ and $t^D \neq \emptyset$. Additionally, the workflow must not have identical XOR/AND branches[2] as well as several creator tasks t_1, t_2 producing the same data output, i.e., $t_1^D = t_2^D$, unless they are contained in different AND branches[2].*

2.3 Taxonomies

The vocabulary of our POCBR system consists of taxonomical representation of terms of the tasks and data objects as previously defined by [17].

Definition 4. *A taxonomy ψ is a tree of a partially ordered set of semantic terms $\Gamma = \{\gamma_0, \ldots, \gamma_n\}$, whereas $\gamma_i \sqsubset \gamma_j$ denotes that γ_j is more general than γ_i (γ_i is more specific than γ_j). Further, $\gamma_i \sqsubseteq \gamma_j$ holds, iff $\gamma_i \sqsubset \gamma_j \vee \gamma_i = \gamma_j$.*

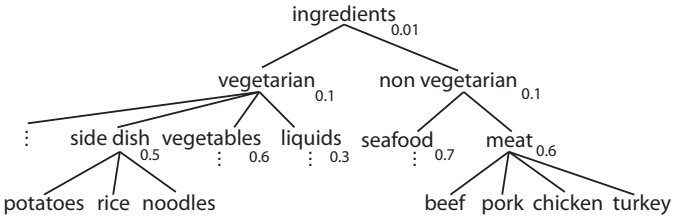

Fig. 3. Example for a data taxonomy

We use two distinct taxonomies, one for the task nodes ψ_{tasks} (preparation steps) and one for the data nodes ψ_{data} (ingredients), i.e., $\Sigma = \Gamma_{\psi_{\text{tasks}}} \cup \Gamma_{\psi_{\text{data}}}$. Hence, the

[1] The terms *completeness* and *consistency* used here with respect to workflows must not be confused with the use of those terms within logics — here, we mean something different, as defined below.

[2] Within the same control-flow block

function S assigns to each node $n \in N$ an appropriate term from $\Gamma_{\psi_{\text{tasks}}}$ or $\Gamma_{\psi_{\text{data}}}$.[3] As an example, figure 3 shows the ingredients taxonomy, in which $beef \sqsubset meat$ holds. Please note that in taxonomies, the leaf nodes represent concrete entities that may occur in executable workflows. For example, a recipe may include potatoes and beef as potential ingredients, but usually not terms from the inner nodes, such as vegetarian or meat. An inner node γ represents a generalized term that stands for the set of more specific terms below it. For example, the generalized term $vegetarian$ stands for the set $\{potatoes, rice, noodles\}$. Further on in the paper we use inner nodes in generalized completion operators to represent that an arbitrary ingredient from the set of its specializations can be chosen. Each inner node γ is additionally annotated with a value sim_ψ defining the similarity between all child terms of γ, (e.g, $sim_\psi(\text{meat}) = 0.6$ in Fig. 3).

3 Workflow Completion Operators

For completing the dataflow of workflows we now introduce completion operators that define valid flows of the data for a particular task. More precisely, the operator defines which input data nodes can be processed by a task and which output data nodes result after the task has been executed. A completion operator is represented as a workflow as defined in Definition 5.

Definition 5. *A completion operator o is a workflow $o = (N_o, E_o)$ consisting of a single task t (also denoted as t_o), i.e., $N_o^T = \{t\}$ and $N_o^C = \emptyset$, together with the set of input data nodes t_D and output data nodes t^D, i.e., $N_o^D = t_D \cup t^D$.*

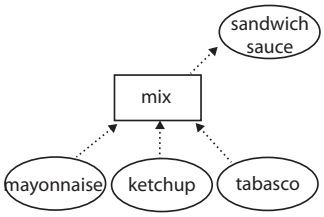

Fig. 4. Example completion operator

An example completion operator is illustrated in Figure 4. It describes that mayonnaise, ketchup, and tabasco can be mixed to produce a sandwich sauce. Operators can also contain generalized terms of tasks and data nodes. Figure 5 shows such a generalized operator. It describes that mayonnaise and any sauce can be combined together with an arbitrary flavoring such that a sandwich sauce is produced.

We now describe under which circumstances an operator is applicable for a task t of workflow w. Therefore, we introduce two mapping functions $m_{in} : t_{oD} \rightarrow N^D$ and $m_{out} : t_o^D \rightarrow N^D$, whose existence is a precondition for the application of an operator

[3] We omit the index if it is obvious which ontology is referenced.

on a task t (see Def. 6). The mappings enforce that the operator data nodes map the current corresponding input and output data nodes of task t. Furthermore, the labels of the operator data nodes must be identical or more general than those labels of the mapped data nodes in the workflow. Furthermore, as m_{in} is a total mapping function, it requires that all input data nodes of the operator o are contained in the workflow w (it is not necessary that they are connected to t). In contrast to m_{in}, m_{out} does not enforce that all output data nodes of the operator are contained in the workflow (in this case the corresponding output is going to be created).

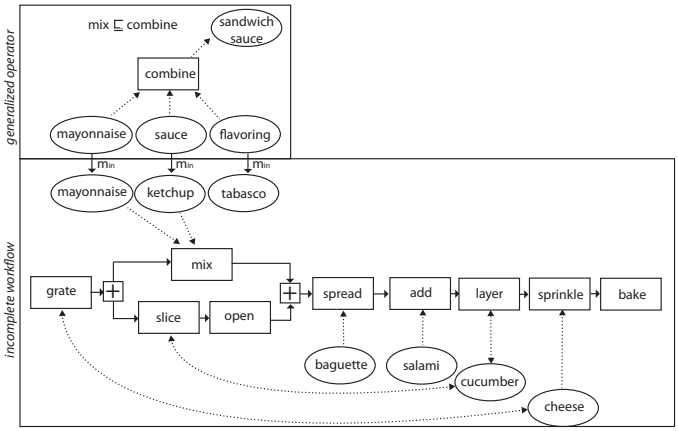

Fig. 5. Example mapping between generalized operator and incomplete workflow

Definition 6. *Let $m_{in} : t_{oD} \rightarrow N^D$ be a total injective mapping for a task t in workflow $w = (N, E)$ s.t. $\forall d \in t_D : \exists d' \in t_{oD} : d = m_{in}(d')$ and $\forall d \in t_{oD} : S(m_{in}(d)) \sqsubseteq S(d)$. Further, $m_{out} : t_o^D \rightarrow N^D$ is a partial injective mapping s.t. $\forall d \in t^D : \exists d' \in t_o^D : d = m_{out}(d')$ and $\forall d \in t_o^D : S(m_{out}(d)) \sqsubseteq S(d)$.*

Based on these mappings we will now explain the applicability of an operator o on task t in Definition 7.

Definition 7. *An operator $o = (N_o, E_o)$ with operator task $o_t \in N_o^T$ is applicable for a task t under the mapping pair (m_{in}, m_{out}) iff $S(t) \sqsubseteq S(o_t)$ and if the mapping functions $m_{in} : t_{oD} \rightarrow N^D$ and $m_{out} : t_o^D \rightarrow N^D$ exist as previously defined.*

Applying a completion operator can result in multiple different workflows, because different data objects might be considered as input or output of a task, i.e., several possible mappings exist. We define a possible workflow resulting from the application of a completion operator with operator task $o_t \in N_o^T$ under (m_{in}, m_{out}) as follows. The mapping describes the input and output data links that have to be constructed in the dataflow of W' if not already present. Additionally, newly added output data nodes also have to be linked to the output of the task t.

Definition 8. *$W' = (N', E')$ is a possible workflow after applying completion operator $o = (N_o, E_o)$ with operator task $o_t \in N_o^T$ under (m_{in}, m_{out}) on the task t of workflow $W = (N, E)$ iff $N' = N \cup N^{D+}$ and $E' = E \cup E^{D+}$. Here, N^+ is the set of output data nodes produced by the operator application, i.e., $N^{D+} = \{d \in o_t^D | \nexists d' \in N^D : S(d') \sqsubseteq S(d)\}$. E^{D+} defines the new input and output data links generated by the operator such that $\forall d \in t_{oD} : (m_{in}(d), t) \in E^{D+}$ and $\forall d \in t_o^D : (t, m_{out}(d)) \in E^{D+}$ and $\forall d \in N^{D+} : (t, d) \in E^{D+}$.*

We will now illustrate the application of a completion operator by an example. Suppose we are going to apply a generalized operator o on task t "mix" of workflow w (see Fig. 5). First, all input data objects of the operator that are not already connected as task input have to be linked. Mayonnaise and ketchup, for example, are already connected as input to task t, but tabasco still has to be linked (see grey marked data link in Fig 6). Finally, the output data object(s) have to be either linked in the same manner or if no matching data object exists for this output data node of the operator N_o^T a new data object is created. In our example, sandwich sauce is created as a data node d in the workflow and linked as an output to the task t (see grey marked workflow elements in Fig 6).

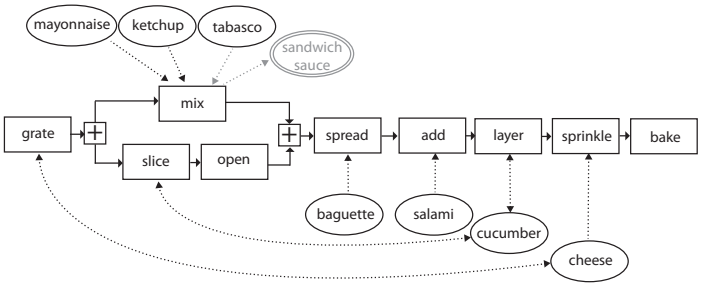

Fig. 6. Example result for an operator application

4 Learning Workflow Completion Operators

To avoid modelling completion operators by hand, we learn them automatically from the workflows from the case base. The idea is that for each complete task t (i.e., $t_d \neq \emptyset$ and $t^d \neq \emptyset$) of workflow $W = (N, E)$ in the case base a corresponding operator $o = (N_o, E_o)$ is constructed, such that t defines the operator task t_o, i.e., $N_o^T = \{t\}$. The data nodes connected to this task $N_o^D = \{d \in N^D | t : d \in t_D \vee d \in t^D)\}$ represent the corresponding input and output data nodes of the operator task t, i.e., $E_o^D = \{(d, t) \cup (t, d) \in E^D | d \in N_o^D\}$.

Next, the learned completion operators are generalized in order to increase their applicability. This means that each task description is generalized to a level of the taxonomy which has a similarity value of at most θ assigned, i.e., $sim_\psi(S(n)) < \theta$. Here, θ is a threshold value that is used to control the degree of generalization performed.

Algorithm COMPLETE_WF(W,O);
Input: Workflow $W = (N, E)$ and completion operators O
Output: Set of completed workflow solutions $SOLUTIONS$
SOL=$\{W\}$;
for each *task* $t \in N^T$ *(tasks ordered by controlflow, i.e, first task is processed first)* **do**
  Init newSOL = $\emptyset$;
  for each *state* $s \in SOL$ **do**
    for each *operator o with mapping pair* $(m_{in}, m_{out}) \in CO(t)$ **do**
      s' := successor state of s by applying o under (m_{in}, m_{out});
      if $\neg$ *(s' can never become consistent)* **then**
        $newSOL = newSOL \cup \{s'\}$;
  SOL=newSOL;
$SOL' = applyDefaultRules(SOL)$;
return $selectCompletedWorkflow(SOL')$;

Algorithm 1: Dataflow completion

When completion operators are constructed from several workflows, we can have several different operators for the same task label, i.e., several different operators for the task "mix". However, a single operator can also be derived from several different workflows. Those operators are stored only once. In addition, a score value $Score(o)$ is recorded that counts how often an operator has been constructed from the case base. This score value reflects a kind of significance, which is used later to prefer the application of highly scored operators during completion.

5 Workflow Completion

We now introduce the approach to complete the dataflow of incomplete workflows. First the learned domain specific operators are applied (Sec. 5.1). Subsequently, we apply some generic default rules (Sec. 5.2) for those tasks that could not be completed appropriately by a domain specific operator.

5.1 Applying learned completion operators

Applying the learned completion operators aims to complete the dataflow of the workflow in such a manner that a best possible completed workflow w.r.t. consistency is produced. The algorithm is based on a breadth first search (see Algorithm 1). Starting from the first task node of the workflow the dataflow is expanded by each *applicable operator* resulting in multiple successor states. For all these successor states the dataflow of the next task w.r.t. the controlflow is expanded by all *applicable operators* and so forth. The set of applicable completion operators for a task t together with the mapping pair under which they are applicable is denoted as $CO(t)$.

In order to reduce run time as well as memory consumption of the algorithm, we withdraw current solutions as soon as it is ensured that they can never become consistent according to Definition 3, i.e., if they contain several creator tasks with identical output

data nodes or identical XOR or AND branches. It turned out that this is sufficient to restrict the search space to a feasible level.

After the application of domain-specific operators a set SOL is created in which for some tasks the completion of the dataflow could be achieved. However, for some tasks no applicable completion operator may exist. Thus, we apply a set of default completion rules on those tasks as described in the next section.

5.2 Applying default completion rules

For all workflow solutions $S = (N, E) \in SOL$ we now apply default completion rules on each task that has no output data nodes, i.e., $t^D = \emptyset$ (see $applyDefaultRules(SOL)$ in Algorithm 1). Thereby, tasks that are incomplete are completed by applying the generic default rules. These default rules represent common, domain independent heuristics for producing a plausible output for a task.

First, from the available set of operators O, we determine which data nodes are used as creator data nodes $D^* = \bigcup_{o \in O} N_o^{D*}$ in the set of operators, i.e., those data nodes which are produced by a task but not consumed by this task, e.g., sandwich sauce. Further, $t^{D-} = \{d \in N^D | \nexists t' \in N^T \wedge d \in t'^D \wedge t' < t\}$ defines the set of unproduced data objects w.r.t. task t, i.e., those data objects that have not been produced by a preceding task t' of t. We refer to such data nodes as unproduced data nodes.

1: **IF** $t^D = \emptyset \wedge \exists! d \in t_D$ **THEN** $t^D := \{d\}$
2: **IF** $t^D = \emptyset \wedge \exists! d \in (t_D \cap N^{D*})$ **THEN** $t^D := \{d\}$
3: **IF** $t^D = \emptyset \wedge \exists! d \in t_D : \exists t' \in N^T \wedge t < t' \wedge d \in t'_d$ **THEN** $t^D := \{d\}$
4: **IF** $t^D = \emptyset \wedge \exists! d \in t^{D-} : \exists t' \in N^T \wedge t < t' \wedge d \in t'_d$ **THEN** $t^D := \{d\}$
5: **IF** $t^D = \emptyset \wedge \exists! d \in t^{D-} : (\exists d^* \in D^* : S(d) = S(d^*)) \wedge (\exists t' \in N^T \wedge t < t' \wedge d \in t'_d)$
THEN $t^D := \{d\}$

Fig. 7. Default rule base ($\exists! x$ denotes that exactly one x exists)

Next, we apply a default rule base for each task t that has no output, i.e., $t^D = \emptyset$, ordered w.r.t. the control-flow (see rule base in Fig. 7). The rule base is applied in a top-down manner, such that only the first applicable rule is fired to determine the output of task t. If t has only one input data object, it is set as the output. Otherwise if t has only one creator data node that exists in the current workflow (i.e. contained in N^{D*}) as an input, it is set as the output. The third rule describes that if exactly one input d of t is used by a successor task as an input again, d is the output. The next rule sets the output of the task t as exactly this data node N^D that has not been produced by a prior task and which is the only one used by the closest possible successor task as an input again. Finally, rule 5 describes the same condition as in rule 4 but restricts the possible output rules to those with labels of the creator data nodes in D^* determined from the available set of operators.

5.3 Heuristical solution selection

After completing the workflow according to the previous sections, we now introduce a heuristical selection method which aims at selecting a single solution from SOL' (see function $selectCompletedWorkflow(SOL)$ in Algorithm 1). As we aim to identify consistent workflows or at least workflows with most completed dataflow we apply the following steps.

1. Initialize SOL' with the set of completed solutions SOL.
2. If SOL' contains at least one consistent workflow, then all other workflows which are not consistent are removed from SOL'.
3. Next, only retain those workflow solutions in SOL' with the largest number of complete tasks, i.e., tasks which have at least one input and one output data node.
4. Finally, rank the remaining workflows in SOL' according to the sum of the score values of the completion operators that have been applied to produce them. Select the workflow with the highest score.

6 Evaluation

The described approach to workflow completion has been implemented as component of the CAKE framework [3]. We investigated, whether the dataflow could be completed such that the result is consistent (Hypothesis H1). Moreover, we validated whether the completed dataflow of the workflows are of acceptable quality (Hypothesis H2).

H1. The presented workflow completion approach constructs consistent workflows in most cases.
H2. The presented workflow completion approach completes the dataflow such a manner that the resulting workflows are of high quality.

6.1 Evaluation Data

We manually constructed 61 sandwich recipe workflows from the textual recipe descriptions from Internet sources with an average size of 20 nodes and 36 edges [16]. All these workflows are consistent, i.e., their dataflow is completely defined and of good quality as the dataflow is constructed according to proper interpretation of the textual recipe. Altogether, they contain 178 different ingredients and 79 tasks. For ingredients and tasks, a taxonomy was manually constructed. The extracted workflows contained AND, XOR, as well as LOOP structures. These 61 workflows serve as our training case base CB_{train}. They further represent golden standard workflows, i.e., as comparison workflows which should be reconstructed by the automatic dataflow completion of the workflows. Previous work (e.g., [18]) showed that such defined consistent workflows are convenient for retrieval and adaptation in POCBR. We performed an ablation study by automatically removing dataflow from the training workflows to construct incomplete test workflows. An investigation of textual recipes showed that for each step the input data objects are usually always explicitly contained in the instructions, while

the output data objects are commonly missing. Thus, we removed all output data links from the workflows, resulting in a set of 61 incomplete workflows as our test case base CB^{I}_{test}. We also generated a second test case base CB^{II}_{test} with a larger deconstruction of the workflow, i.e., we constructed workflows such that they only consist of input data nodes and only input data links to this task by which the corresponding input data is consumed for the first time w.r.t. the control-flow. Consequently, only the ingredients contained in the ingredient list are contained in the workflows' dataflow. As quality measures of an automatically completed workflow and a golden standard workflow we employ the precision and recall measures as defined by Schumacher et al. [22].

$$precision_T(t, t^*) = \frac{|t^D \cap t^{*D}| + |t_D \cap t^*_D|}{|t^D| + |t_D|} \tag{3}$$

$$recall_T(t, t^*) = \frac{|t^D \cap t^{*D}| + |t_D \cap t^*_D|}{|t^{*D}| + |t^*_D|} \tag{4}$$

$Precision_T$ defines the percentage of the data links of task t that are contained in the golden standard task t^*, while $recall_T$ defines the percentage of data links of the golden standard task t^* that have been reconstructed.

$$precision(W, W^*) = \frac{1}{|N^T|} + \sum_{i=0}^{|N^T|} precision_t(t_i, t^*_i) \tag{5}$$

$$recall(W, W^*) = \frac{1}{|N^T|} + \sum_{i=0}^{|N^T|} recall_t(t_i, t^*_i) \tag{6}$$

$Precision$ (see Eq. 5) defines how many of the tasks' data links are contained in the golden standard workflow W^*, while recall (see Eq. 6) defines how many of the tasks' data links of the golden standard workflow W^* have been reconstructed. In both equations t_i and t^*_i refer to the same task but w.r.t. reconstructed workflow and golden standard workflow respectively.

$$F1(W, W^*) = 2 \cdot \frac{precision(W, W^*) \cdot recall(W, W^*)}{precision(W, W^*) + recall(W, W^*)} \tag{7}$$

The F1 measure aggregates both, the precision and recall values in a single score, which is basically the harmonic mean of precision and recall.

6.2 Experimental Evaluation and Results

We employed the leave-one-out evaluation principle to evaluate our presented approach. More precisely, we selected each workflow w as a test workflow from the case base CB^{I}_{test} or CB^{II}_{test} respectively. Next, we learned all operators from the training case base CB_{train} (without the corresponding training workflow of test workflow w) resulting in a set of operators O_w. In average for each of the test workflows 283 operators have been generated.

Next we applied our presented approach by completing the dataflow of workflow w using the set of operators O_W learned from the other workflows. In average 6.41 completion operators for CB^I_{test} and 5.26 completion operators for CB^{II}_{test} have been applied for each workflow. In average 2.0 solutions in SOL' have been generated by CB^I_{test} and 3.0 solutions in SOL' by CB^{II}_{test} of which the presented heuristic (see Sec. 5.3) identified a single solution. After completing the dataflow of the workflow, 44 test workflows (72.13%) have been transformed to consistent workflows for CB^I_{test}. However, for the test case base CB^{II}_{test} only 4 test workflows (6.56%) have been transformed to consistent workflows. Thus, Hypothesis H1 has been mostly confirmed for CB^I_{test} but not for CB^{II}_{test}. This shows that consistent solutions can only be constructed if enough information either in the incomplete workflow description or in the learned completion operators is contained.

Table 1. Quality compared to golden standard workflow

	$\boldsymbol{CB^I_{test}}$		$\boldsymbol{CB^{II}_{test}}$	
	uncompleted case	completed case	uncompleted case	completed case
precision	1.00	0.98 (-0.02)	1.00	0.85 (-0.15)
recall	0.59	0.98 (+0.39)	0.46	0.74 (+0.28)
F1	0.74	0.98 (+0.24)	0.63	0.79 (+0.16)

Moreover, we investigated the quality of the completed workflow by comparing it to the corresponding training workflow w_{train}. The results are illustrated in Table 1 showing the average precision, recall, and F1 score values for CB^I_{test} and CB^{II}_{test}. We can see that while the precision has not been significantly reduced (low rate of wrong data links) the recall has significantly been increased from 0.6 to 0.98 resulting also in a weighty increase of the F1 score. Further, Table 1 shows for CB^{II}_{test} that the recall (+0.27) and F1 score (+0.16) have significantly been increased. In contrast to CB^I_{test} the precision score (-0.15) indicates that data links are more likely not in alignment with the golden standard workflow. This implies that the quality of the completion significantly relies on the available knowledge in the incomplete case. However, in total hypothesis H2 is confirmed as the F1 score has been significantly increased in both scenarios.

7 Conclusions and Future Work

We presented a novel approach for the completion of workflow cases which lessens the knowledge gap in the case base and increases the capabilities of POCBR applications. The evaluation showed that if the knowledge gap in the cases is small the completion algorithm is able to identify consistent solutions. Previous work has showed that such consistent workflows are convenient for retrieval and adaptation in POCBR [18]. Further, the quality of the completed dataflow is high even if the knowledge gap in the workflow case description is larger.

Various approaches have been presented for case completion in CBR [6,1,10,20]. However, in POCBR case completion has only been marginally investigated so far.

Schumacher [21] as well as Dufour-Lussier et al. [9], for example, investigated the automatic extraction of workflows from textual process descriptions. As such texual descriptions usually lack of information, both employ linguistic methods to complete workflow cases. Workflow completion has been also considered for Workflow Management Systems (WFMS) with the goal to complete the workflows such that they become executable [26,12].

Future work will comprise an extended evaluation towards other domains and further varying characteristics of the case base. Moreover, we plan to investigate methods in case of inefficient run time or memory consumption, e.g., for larger workflows or less completed workflows. Finally, we aim to increase the case completion of workflows with regard to consistency using additional default completion rules and by abstracting the learned completion operators.

Acknowledgements. This work was funded by the German Research Foundation (DFG), project number BE 1373/3-1.

References

1. Bach, K., Althoff, K.D., Satzky, J., Kroehl, J.: CookIIS Mobile: A Case-Based Reasoning Recipe Customizer for Android Phones. In: Petridis, M., Roth-Berghofer, T., Wiratunga, N. (eds.) Proceeding of UKCBR 2012, December 11, Cambridge, United Kingdom. pp. 15–26. School of Computing, Engineering and Mathematics, University of Brighton, UK (2012)
2. Badra, F., Cordier, A., Lieber, J.: Opportunistic Adaptation Knowledge Discovery. In: McGinty, L., Wilson, D.C. (eds.) Case-Based Reasoning Research and Development, ICCBR 2009, Seattle, WA, USA, Proceedings. LNCS, vol. 5650, pp. 60–74. Springer (2009)
3. Bergmann, R., Gessinger, S., Görg, S., Müller, G.: The collaborative agile knowledge engine CAKE. In: Proceedings of the 18th International Conference on Supporting Group Work. pp. 281–284. ACM (2014)
4. Bergmann, R., Gil, Y.: Similarity assessment and efficient retrieval of semantic workflows. Inf. Syst. 40, 115–127 (2014)
5. Bergmann, R., Wilke, W., Vollrath, I., Wess, S.: Integrating General Knowledge with Object-Oriented Case Representation and Reasoning. In: Burkhard, H.D., Wess, S. (eds.) 4th German Workshop: Case-Based Reasoning - System Development and Evaluation. pp. 120–126. Humboldt-Universität Berlin, Informatik-Berichte Nr. 55 (1996)
6. Burkhard, H.: Case completion and similarity in case-based reasoning. Comput. Sci. Inf. Syst. 1(2), 27–55 (2004)
7. Craw, S., Wiratunga, N., Rowe, R.: Learning adaptation knowledge to improve case-based reasoning. Artif. Intell. 170(16-17), 1175–1192 (2006)
8. Davenport, T.: Process Innovation: Reengineering Work Through Information Technology. Harvard Business Review Press (2013)
9. Dufour-Lussier, V., Ber, F.L., Lieber, J., Nauer, E.: Automatic case acquisition from texts for process-oriented case-based reasoning. Inf. Syst. 40, 153–167 (2014)
10. Hanft, A., Schfer, O., Althoff, K.D.: Integration of Drools into an OSGI-based BPM-Platform for CBR. In: Minor, M., Montani, S., Recio-Garcia, J.A. (eds.) Workshop on Process-Oriented Case-Based Reasoning. ICCBR 2011, 19th, September 12-15, Greenwich, London, United Kingdom. University of Greenwich (2011)

11. Hanney, K., Keane, M.T.: The Adaption Knowledge Bottleneck: How to Ease it by Learning from Cases. In: Leake, D.B., Plaza, E. (eds.) Case-Based Reasoning Research and Development, ICCBR-97, Providence, USA. LNCS, vol. 1266, pp. 359–370. Springer (1997)
12. Koehler, J., Tirenni, G., Kumaran, S.: From Business Process Model to Consistent Implementation: A Case for Formal Verification Methods. In: Proceedings of EDOC 2002, 17-20 September 2002, Lausanne, Switzerland. p. 96. IEEE Computer Society (2002)
13. Minor, M., Görg, S.: Acquiring Adaptation Cases for Scientific Workflows. In: Case-Based Reasoning. Research and Development, ICCBR 2011. LNCS, vol. 6880, pp. 166–180. Springer (2011)
14. Minor, M., Montani, S., Recio-Garca, J.A.: Process-oriented case-based reasoning. Information Systems 40(0), 103 – 105 (2014)
15. Müller, G., Bergmann, R.: Workflow Streams: A Means for Compositional Adaptation in Process-Oriented CBR. In: Lamontagne, L., Plaza, E. (eds.) Case-Based Reasoning Research and Development, ICCBR 2014, Cork, Ireland, September 29, 2014 - October 1, 2014. Proceedings. LNCS, vol. 8765, pp. 315–329. Springer (2014)
16. Müller, G., Bergmann, R.: CookingCAKE: A Framework for the Adaptation of Cooking Recipes Represented as Workflows. In: Kendall-Morwick, J. (ed.) Workshop Proceedings, ICCBR 2015, Frankfurt, Germany, September 28-30, 2015. CEUR Workshop Proceedings, vol. 1520, pp. 221–232. CEUR-WS.org (2015)
17. Müller, G., Bergmann, R.: Generalization of Workflows in Process-Oriented Case-Based Reasoning. In: Russell, I., Eberle, W. (eds.) Proceedings, FLAIRS 2015, Hollywood, Florida. May 18-20, 2015. pp. 391–396. AAAI Press (2015)
18. Müller, G., Bergmann, R.: Learning and Applying Adaptation Operators in Process-Oriented Case-Based Reasoning. In: Hüllermeier, E., Minor, M. (eds.) Case-Based Reasoning Research and Development, ICCBR 2015, Frankfurt am Main, Germany, September 28-30, 2015, Proceedings. LNCS, vol. 9343, pp. 259–274. Springer (2015)
19. Richter, M.M., Weber, R.O.: Case-Based Reasoning - A Textbook. Springer (2013)
20. Roth-Berghofer, T.: Explanations and Case-Based Reasoning: Foundational Issues. In: Funk, P., González-Calero, P.A. (eds.) Advances in Case-Based Reasoning, ECCBR 2004, Madrid, Spain, Proceedings. LNCS, vol. 3155, pp. 389–403. Springer (2004)
21. Schumacher, P.: Workflow Extraction from Textual Process Descriptions. Dissertation, Johann Wolfgang Goethe Universität Frankfurt/Main (2016)
22. Schumacher, P., Minor, M., Schulte-Zurhausen, E.: Extracting and enriching workflows from text. In: IEEE 14th International Conference on Information Reuse & Integration, IRI 2013, San Francisco, CA, USA, August 14-16, 2013. pp. 285–292. IEEE (2013)
23. Schumacher, P., Minor, M., Walter, K., Bergmann, R.: Extraction of procedural knowledge from the web: a comparison of two workflow extraction approaches. In: Mille, A., Gandon, F.L., Misselis, J., Rabinovich, M., Staab, S. (eds.) Proceedings of WWW 2012, Lyon, France, April 16-20, 2012 (Companion Volume). pp. 739–747. ACM (2012)
24. Stahl, A.: Learning of Knowledge-Intensive Similarity Measures in Case-Based Reasoning. Dissertation, Technische Universität Kaiserslautern (2004)
25. Thaler, T., Dadashnia, S., Sonntag, A., Fettke, P., Loos, P.: The IWi Process Model Corpus. Tech. rep., Institute for Information Systems (IWi) at the German Research Center for Artificial Intelligence (DFKI) (10 2015)
26. Trcka, N., van der Aalst, W.M.P., Sidorova, N.: Workflow completion patterns. In: IEEE Conference on Automation Science and Engineering, CASE 2009, Bangalore, India, 22-25 August, 2011. pp. 7–12. IEEE (2009)
27. Wilke, W., Vollrath, I., Bergmann, R.: Using Knowledge Containers to Model a Framework for Learning Adaptation Knowledge. In: Wettschereck, D., Aha, D.W. (eds.) ECML 1997. MLNet Workshop Notes - Case-Based Learning: Beyond Classification of Feature Vectors. pp. 68–75. NCARAI, Washington, D. C., USA (1997)