Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

TIDY: A PBE-based framework supporting smart transformations for entity consistency in PowerPoint

Shuguan Liu, Huiyan Wang *, Chang Xu **

State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China
Department of Computer Science and Technology, Nanjing University, Nanjing, China

ARTICLE INFO

Keywords:
Programming by example
Rich-formatted documents

ABSTRACT

Context: Programming by Example (PBE) is increasingly assisting human users by recognizing and executing repetitive tasks, such as text editing and spreadsheet manipulation. Yet, existing work falls short on dealing with rich-formatted documents like PowerPoint (PPT) files, when examples are few and collecting them is intrusive.

Objective: This article presents TIDY, a PBE-based framework, to assist automated entity transformations for their layout and style consistency in rich-formatted documents like PowerPoint, in a way adaptive to entity contexts and flexible with user selections.

Methods: TIDY achieves this by examining entities' operation histories, and proposes a two-stage framework to first identify user intentions behind histories and then make wise next-operation recommendations for users, in order to maintain the entity consistency for rich-formatted documents.

Results: We implemented TIDY as a prototype tool and integrated it into PowerPoint as a plug-in module. We experimentally evaluated TIDY with real-world user operation data. The evaluation reports that TIDY achieved promising effectiveness with a hit rate of 77.3% on average, which was stably holding for a variety of editing tasks. Besides, TIDY took only marginal time overhead, costing several to several tens of milliseconds, to complete each recommendation.

Conclusion: TIDY assists users to complete repetitive tasks in rich-formatted documents by non-intrusive user intention recognition and smart next-operation recommendations, which is effective and practically useful.

1. Introduction

Internetware applications are featured by context-awareness and smart adaptation. Programming by examples (PBE) techniques are enabling Internetware applications by learning from contexts (users' inputs) and making required adaptations to substitute human repetitive actions (automating task executions by synthesized operations). PBE [1], as an emergent and promising sub-field of program synthesis [2], can free people with no programming background from those tedious and repetitive tasks. Given some input–output examples as the specification, a PBE technique can synthesize a program that satisfies the specification and also generalizes well to new inputs. PBE has been widely applied in many application domains such as data wrangling/transformation [3,4] and code transformation [5,6].

However, there are limitations of typical PBE work on rich-formatted documents. On one hand, many PBE systems require users to enter a special mode to provide examples, and this could interrupt users' normal workflows and increase unexpected workloads. On the other hand, although a specification consisting of useful examples could sometimes be available from users, it can still lead to ambiguity, since there could be multiple synthesized programs that "seemingly" satisfy these examples. Therefore, in order to better synthesize an intended program, it may still require a certain amount of high-quality examples. Although some PBE work [7] might require seemingly only several examples for specific cases, this achievement could restrict to certain scenarios (e.g., string manipulation), in which the search space itself may not be large and a few examples could already suffice. However, for other complex scenarios, the work's underlying machine learning mechanism may need more examples for a robust training. This requirement can more than what can be afforded for rich-formatted documents, e.g., PowerPoint, for the reason that therefore could be only several entities aiming for the same operations in a page, while spreadsheets can have much more (e.g., several tens or even hundreds of) cells that carry the same computational tasks.

* Corresponding author at: Department of Computer Science and Technology, Nanjing University, Nanjing, China.
** Corresponding author at: Department of Computer Science and Technology, Nanjing University, Nanjing, China.
E-mail addresses: liu_shuguan@126.com (S. Liu), cocowhy1013@gmail.com (H. Wang), changxu@nju.edu.cn (C. Xu).

https://doi.org/10.1016/j.infsof.2021.106611
Received 29 October 2020; Received in revised form 15 March 2021; Accepted 29 April 2021

For the above problems, this article proposes the following solution TIDY: one does not have to ask users to provide examples in a special mode; instead, users' intentions (tasks to be completed) are identified automatically through users' history operations and their contexts; then suggestions of recommended operations can be generated for relevant entities for completing the intended task. Besides, to be flexible, rather than pursuing the only intention, we suggest operations of several possible user intentions and then gradually figure out the exact intention when analyzing more user operations at runtime. Moreover, concerning a specific rich-formatted document, to restrict the search space and make its recommendation more accurate, TIDY would be also integrated with a corresponding domain-specific goal library for modeling common user intentions.

To be specific, our TIDY approach proposes a two-stage framework to automatically maintain the entity consistency for rich-formatted documents. First, it would automatically identify user intentions from any given user operation history, aiming to obtain a clear and executable user intention behind the history. Second, based on such identified user intention, TIDY would scan the whole document and generate possible user next-operations for recommendation with prioritization. Those recommended next-operations are expected to automatically assist users to maintain entity consistency at runtime instead of the original repetitive editing by users themselves without our TIDY. As such, this article makes the following contributions: a domain-agnostic framework providing recommendations for subsequent entities and operations, and its specific technical implementation in the domain of popular PowerPoint application.

To evaluate TIDY's performance, we implemented TIDY as a prototype toolkit and integrated it as a plug-in module into PowerPoint. We evaluated TIDY's effectiveness on recommending users' true next-operations by hit rate and time overhead. We observe that for a total of 2363 collected real-world user histories from 21 participants, TIDY achieved a promising hit rate 77.3% on average with its default setting, suggesting its general effectiveness. Besides, TIDY's effectiveness consistently held across different factor settings. On the other hand, TIDY's time overhead was only several to several tens of milliseconds per instance (7.2 ms on average), which is marginal and acceptable for runtime operations, suggesting its practical usefulness.

The remainder of this article is organized as follows. Section 2 presents background knowledge for our target problem of maintaining the entity consistency for rich-formatted documents. Section 3 introduces necessary notions, and based on them elaborates on our TIDY approach on identifying user intentions and making recommendation for achieving the entity consistency. Then, Section 4 explains how to apply TIDY to one of the most popular rich-formatted document application, PowerPoint, and based on it, Section 5 evaluates TIDY's performance on both its effectiveness and efficiency in details. Section 6 discusses some issues concerning TIDY's usage. After that, Section 7 discusses the related work in recent years, and Section 8 concludes this article.

2. Background

In this section, we first introduce some background knowledge of PBE, and then present an example on maintaining entity consistency for motivating our work.

2.1. PBE background

Programming by Example (PBE) is a popular technique to help automatically generate programs from given examples of input–output pairs. Formally, given a set of examples $\{p_1, p_2, \ldots, p_n\}$, each element of which refers to an input–output pair like $p_i = \langle input_i, output_i \rangle$, a PBE technique would generate a program P that when fed by any existing input $input_i$ of given examples, would produce its corresponding output $output_i$ correctly. Moreover, the logics in this generated program is expected to be generalizable to other similar examples. Usually, in order to obtain an expected program P, a PBE technique would require a certain amount of examples in order to guide its space searching for a proper program P. Due to its nice superiorities on automatic input–output transformations, PBE techniques have been successfully applied to applications with great structural examples, e.g., spreadsheets [4,8], file management [9], and data parsing and extraction [3].

However, for our targeted rich-formatted documents in this article, only few PBE research [10] has been conducted for them, and there are obvious challenges that: (1) rich-formatted documents usually contain not enough examples for PBE's program synthesis, and (2) user intentions behind such formatted examples can be relatively more subtle than value examples like spreadsheet cells. Our work specifically targets at this problem, and aims to maintain entity consistency for rich-document applications by a PBE-based framework, which derives and instantiates clear user intentions from a few formatted examples, and then instead of giving concrete synthesized programs, makes multiple operation recommendations for users to choose from, which can also support to adaptively evolve at runtime.

2.2. Motivating example

We give a motivating example in Fig. 1 to illustrate how entity consistency should be maintained in rich-formatted documents like PowerPoint. Fig. 1 gives an example illustrating with editing history in (a) and expected operations in (b) for entity consistency with PowerPoint-alike pages. In Fig. 1(a), there are a total of nine entities on the page, each of which refers to a colored rectangle (colored in blue or purple).

Suppose a user has edited this page and moved entities e_1 and e_3 in history to be bottom-aligned, as shown by the two arrows. The entities' original locations before moving were drawn by dashed rectangles for ease of illustration. To make the entity consistency, a desired approach is expected to identify the possible user intention behind its e_1 and e_3 movements, i.e., "moving blue rectangles to be bottom-aligned", and accordingly make suggested movements exactly as shown in Fig. 1(b), i.e., moving e_6, e_7, and e_9 to be bottom-aligned as well. Note that, in such a formatted document, there are only two examples, with each operation denoting a natural PBE example of the associated entity's original state as input and its present state as output, e.g., $\langle e'_1, e_1 \rangle$ and $\langle e'_3, e_3 \rangle$.

This can hardly meet the requirement of most normal PBE systems to effectively synthesize its required program to conduct such operations due to their large program search space. Although some PBE work [7] seems to only require several examples for specific cases in recommendation, its kernel machine learning mechanism still asks for quite plenty of examples for robust training when it comes to scenarios other than certain scenarios (e.g., string manipulation), in which the search space itself may not be large and a few examples could already suffice. This could be more than what can be afforded for rich-formatted documents, e.g., PowerPoint. The reason is that, unlike spreadsheets having hundreds or even thousands of cells that need to complete the same computational task, in a PowerPoint page, there could be only several entities that need to do so.

Our TIDY approach would gradually derive a few different user intentions from history operations, i.e., moving e_1 and e_3 with the aid of domain-specific knowledge. Although by only analyzing e_1's movement, TIDY may identify some other user intentions like "moving blue rectangles downward by distance a", it would be naturally adjusted or discarded when TIDY further analyses e_3's movement. By doing so, TIDY can identify and present the most suitable user intentions behind the collected movement history, e.g., "moving blue rectangles to be bottom-aligned". After that, TIDY would search the whole document for similar entities to existing moved entities, and suggest to apply similar operations to complete this certain user intention. In this case, TIDY would easily find all remaining blue entities (i.e., e_6, e_7, and e_9), clearly

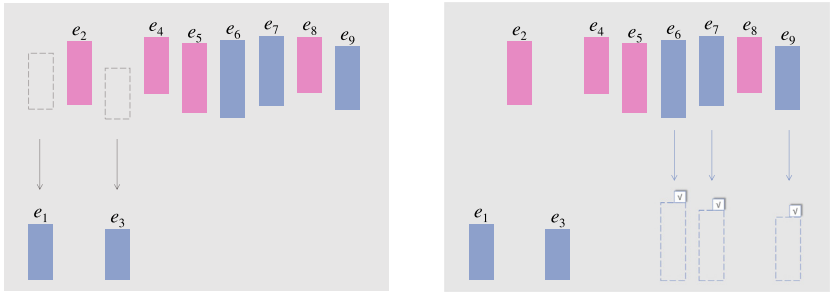

Fig. 1. The motivating example. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

sharing the same color with existing e_1 and e_3, and then suggest to move them as TIDY's recommended next-operations for users to choose from, as the check marks in Fig. 1(b). Instead of users' repetitive operations by moving all e_6, e_7, and e_9 carefully to be bottom-aligned, one only needs to make some clicks now. In the following, we would elaborate on the details of our TIDY approach.

3. Methodology

In this section, we elaborate on our approach TIDY and explain how to apply it to achieve entity consistency (e.g., layout and style consistency) for rich-formatted documents, e.g., PowerPoint.

3.1. Overview

As we mentioned before, TIDY aims to understand user intentions and effectively assist users through wise next-operation recommendation to achieve entity consistency, which usually requires repetitive and exhaustive user operations. To do so, we divide this problem into two parts. First, how to automatically identify the user intention from obtained user operations in history, and second, how to, based on such identified user intentions, make a wise next-operation recommendation accordingly. We give an overview in Fig. 2. TIDY consists of two corresponding stages and each stage uses two steps to achieve one problem part as mentioned earlier.

In the first stage (User Intention Identification), in order to identify the user intention, TIDY proposes the notion of *goal* for modeling user intentions. To be specific, a *goal* is first selected from a prepared *goal library* customized for a certain rich-formatted document type in TIDY's application, which is general and abstract with parameters in the library at the beginning (i.e., a *parameterized goal* for simply denoting a quite rough user intention direction), e.g., "make entities in a uniform color x". Then, it would be gradually instantiated during TIDY's analyses on obtained user operations in history (i.e., an *instantial goal* for identifying a specific user intention for analyzed operations), e.g., an instantial version of the former example probably being "make entities in a uniform color of **red**". After that, TIDY can eventually identify instantial goals (parameter-free) for the analyzed operation history in this stage, which denote clear user intentions and would later be fed into the second stage for the coming next-operation recommendation.

In the second stage (Next-operation Recommendation), TIDY would scan the document to identify some candidate entities *relevant* to those identified instantial goals associated with the given operation history for recommendation through entity relevance calculation, and then generate operations upon those entities for meeting the obtained instantial goals as next-operation recommendations with prioritization.

In the following, we first present some necessary notations and definitions, and then elaborate on the TIDY approach in detail.

3.2. Notations and definitions

Entity. An entity refers to a piece of object associated in rich-formatted documents [10]. In this article, we model an entity as a finite set of key–value pairs representing its state information, each of which specifies an entity attribute and its associated attribute value, i.e., entity $e = \{attribute_1 : value_1, \dots, attribute_m : value_m\}$. For illustration, in a popular rich-formatted document PowerPoint, a drawn rectangle entity can be modeled as a key–value set of a rectangle object's associated attributes, e.g., Height, Width, etc. Different entities may be associated with different attributes and attribute values.

Operation. An operation refers to a user manipulation relating to attributes in a specific entity. We model an operation by specifying its targeted entity for manipulation and attributes with expected values of this operation. For example, an operation only to change entity e_k's attributes $attribute_i$ to $value'_i$ and $attribute_j$ to $value'_j$ can be modeled as $op = \langle e_k, (attribute_i : value'_i, attribute_j : value'_j)\rangle$. In this case, e_k is op's targeted entity, and straightforwardly, entity e_k after this operation would become: $e_k = \{attribute_1 : value_1, \dots, attribute_i : value'_i, \dots, attribute_j : value'_j, \dots, attribute_m : value_m\}$, with only $attribute_i$ and $attribute_j$ being changed in their values.

Goal. A goal refers to a user intention in manipulating objects. We model a goal as manipulation targets of interesting attributes and expressional descriptions for manipulation. For example, a goal "only change the entity width to make an entity right-aligned" can be modeled as: $\langle(attribute_{width}), (value_{left} + value_{width} == x)\rangle$, while x represents a specific right-aligned location. For ease of presentation, we call those attributes interesting to a goal to be this goal's *goal-targeted attributes*. This is a typical *parameterized goal*, with parameters in its expression, representing a general user intention but still waiting to be instantiated. If all parameters in a parameterized goal has been initialized with specific values, we call it an *instantial goal*, and this process *goal instantiation*. For example, $\langle(attribute_{width}), (value_{left} + value_{width} == 100)\rangle$ is a instantial goal, namely $goal_1$, which has no parameter that has not been initialized yet, representing a clear and executable user intention for "only change the entity width to make an entity right-aligned to location 100". Moreover, such an instantial goal can directly guide how to manipulate an entity for meeting this goal, easily producing operations for recommendation. For example, assuming entity e_k with $attribute_{left}$ and $attribute_{width}$ being 80 and 5 at present, in order to meet $goal_1$, a natural manipulation is to change the goal-targeted attribute $attribute_{width}$ from 5 to 20, so that $value_{left} + value_{width} == 100$ in $goal_1$ would be satisfied. In this case, the operation for recommendation (i.e., *next-operation*) is $op = \langle e_k, (attribute_{width} : 20)\rangle$.

3.3. Stage 1: User intention identification

In this Stage, TIDY analyzes user operations in history for user intention identification, based on a parameterized goal library customized for the target documents TIDY is applied to. To be specific, this stage would first try to select *related* parameterized goals as candidates from

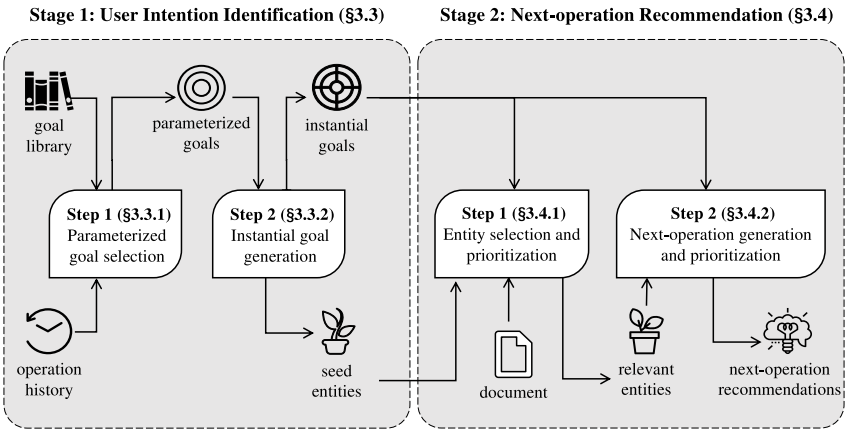

Fig. 2. TIDY overview.

the library based on analyzing obtained user operations (step 1), and then instantiate those selected parameterized goals (step 2) in order to better support the latter next-operation recommendation.

3.3.1. Step 1: Parameterized goal selection

Assume all parameterized goals in the library to be $g_1, g_2, \ldots, g_m$, and obtained user operations in history to be $op_1, op_2, \ldots, op_n$. TIDY regards a goal to be *related* to an operation if this operation changes any of the goal's goal-targeted attributes. Therefore, given any user operation, it is straightforward to identify its related goals from the library. Then, TIDY proposes a backward relation analysis from op_n to op_1, to identify their related goals individually, as detailed in Algorithm 1. Since we observe that for a specific user intention, its associated user operations tend to be continuous, TIDY tracks related goals backward from op_n to op_1, and selects only the goals that continuously exhibit related from the "beginning" (op_n) as goal candidates for selection. Meanwhile, those specific continuous operations relating to each goal candidate are its goal-related operations. For example, if goal g_k relates to all operations from op_n back to op_i, with op_{i-1} unrelated, then the goal is one goal candidate for selection and operations op_n to op_i are its goal-related operations. To be specific, each goal's goal-related operations $(op_i, \ldots, op_n)$ actually compose a continuous operation subsequence of the original user operation history $(op_1, op_2, \ldots, op_n)$ with a necessary op_n. Note that, considering that users may possibly not stick to one intention due to unexpected disturbance (e.g., jumping to some irrelevant actions unexpectedly), some unexpected operations may occur in the middle of a collected sequence and somehow make a goal's associated operations non-consecutive. To alleviate possible problems, we adopted a tolerance treatment in TIDY to allow a few unexpected operations occurring (i.e., causing a non-consecutive or noisy sequence) when handling a sequence of user operations for a specific goal. That is, in Algorithm 1, we allow a few times matching failures with the setting of a *budget* variable, which is initialized by a fed value of tolerance size. To be specific, now in this algorithm, every time a matching failure occurs (intersection being empty at Line 11), it would reduce the budget variable by one (Line 12), and only when the variable value becomes zero, TIDY would break the loop and stop the analysis. Note that, with this tolerance treatment, only consecutive matching failures would be counted in accumulation since the budget variable would be reset each time a matching success occurs (Line 10). For example, when the fed tolerance size is set to be two initially in Algorithm 1, only TIDY meeting three consecutive matching failures would lead to an analysis stop. In this way, the non-consecutive operation problem can be alleviated. Detailed investigations about how different tolerance sizes would affect TIDY's effectiveness would be later discussed in Section 5.3.2.

Until now, all selected goals are still parameterized, which cannot be directly used for generating clear next-operations in recommendation. In order to help better conduct next-operations recommendation and avoid confusions, TIDY in the following proposes to instantiate them to obtain a clear and executable instantial goal for a clearer user intention.

Algorithm 1: Parameterized Goal Selection

```
1  Function Parameterized Goal Selection(H, GL, tolerance_size):
       // H : [op_1, ..., op_n], GL:the goal library
2      foreach pg in GL do
3          if H[H.len − 1].changed_attrs ∩ pg.targeted_attrs == ∅ then
4              continue;// must be op_n's related goal
5          related_op_seq = [H[H.len − 1]];// necessary op_n
6          budget = tolerance_size
7          for k ← H.len−2 to 0 do
8              if H[k].changed_attrs ∩ pg.targeted_attrs! = ∅ then
9                  insert H[k] into the front end of related_op_seq
10                 budget = tolerance_size
11             else
12                 if − − budget < 0 then
13                     break
14         step1_output.add(< pg, related_op_seq >) // empty at first
15     return step1_output;
```

3.3.2. Step 2: Instantial goal generation

In the last step, for any given operation history in sequence, $op_1, \ldots, op_n$, TIDY selects the sequence's related parameterized goals from the library and obtains each selected parameterized goal's associated goal-related operations. Each parameterized goal actually denotes a general and rough user intention for its goal-related operations. Then, in this step, we introduce how TIDY instantiates a parameterized goal with its goal-related operations to make the goal's inner user intention clearer and executable.

Let a parameterized goal be g_s in selection and its goal-related operations be $op_i, \ldots, op_n$. To do so, TIDY instantiates goal g_s by analyzing its goal-related operations in a backward order. Starting from op_n, TIDY puts actual attribute values of this operation's targeted entity into the parameterized expressions of goal g_s, thus trying to assign concrete values to its referred attributes. For example, as aforementioned, for a parameterized goal $g_e = \langle (attribute_{width}), (value_{left} + value_{width} == x) \rangle$, suggesting a user intention of "only change the entity width to be right-aligned", let the analyzed operation at the moment from its goal-related operations be $op_e = \langle e_k, (attribute_{width} : 80) \rangle$ and e_k be $\{attribute_{left} : 20, \ldots, attribute_{width} : 40\}$ originally before this operation. Then, when

TIDY instantiates this parameterized goal g_e with op_e, it assigns to all attribute values in g_e's parameterized expression with the attribute values of this operation's targeted entity (i.e., e_k in this case), which are updated if requested. That is, in g_e's parameterized expression ($value_{left} + value_{width} == x$), $value_{left}$ is assigned with $attribute_{left}$'s original value 20, and $value_{width}$ is assigned with $attribute_{width}$'s updated value 80 since op_e has just changed it from 40 to 80 at the moment to meet goal g_e. Therefore, with the expression now being $20 + 80 == x$, it is natural to instantiate g_e's parameter x with value 100 and transform parameterized goal g_e into an instantial goal $g_e = \langle(attribute_{width}), (value_{left} + value_{width} == 100)\rangle$.

We present details in Algorithm 2. For a parameterized goal g_s in selection and its goal-related operations $op_i, \ldots, op_n$, TIDY instantiates goal g_s by analyzing its goal-related operations in a backward order (Line 8). Considering each analyzed operation, parameterized expressions in g_s would be instantiated with latest attribute values of this operation's targeted entity as aforementioned and then stored (Line 9). If not all attribute values in g_s's parameterized expressions can be instantiated by an operation, we regard this operation to be not practically related to this goal and stop the analysis process (Line 10). In every loop iteration, TIDY tries to solve stored expressions, following a typical expression solve process [11] and then assign concrete values to g_s's parameters (Line 11), e.g., x in g_e as mentioned before. Otherwise, TIDY regards this goal to be not suitable yet for the next-operation recommendation (Line 18), since its intention cannot be identified clearly through its related user operations so far, including both unsolvable (i.e., loop break (Line 20)) and multiple solution cases (loop continue (Line 22)).

Algorithm 2: Instantial Goal Generation

```
1  Function Outer Loop(step1_output):
       // Entrance for instantiating parameterized goals
2      foreach < para_goal, related_op_seq > in step1_output do
3          flag, tmp = Instantiate Goal(para_goal, related_op_seq);
4          if flag then
5              step2_output.add(tmp);// Successfully instantiated
6      return step2_output;
7  Function Instantiate Goal(pg, RH):
       // Goal instantiation for a parameterized goal pg
8      for k ← RH.len − 1 to 0 do
9          try E = pg.instan_attr_values(RH[k]);
10         catch return <0, null>;
11         < state, solution > = Solve(ES.add(E));
12         if state == 1 then
13             ig = pg.instan(solution); // solution→instantial
                   goal
14             seed_entities ← (RH[k].ori_entity ...
               RH[RH.len-1].ori_entity);
15             while k − 1 ≥ 0 and RH[k = k − 1] can match ig do
16                 seed_entities.add(RH[k].ori_entity);
17             return < 1, < ig, seed_entities >>;// unique solution
18         else
19             if state == 0 then
20                 break;// no solution, drop the goal
21             else
22                 continue;// state == −1, multiple solutions
23     return <0, null>;// still cannot have unique solution
24 Function Solve(ES):
25     ... // solve expressions stored in ES
26     if solution_num == 1 then return <1, solution> ;
27     if solution_num == 0 then return <0, null> ;
28     else return <-1, null>;
```

For ease of the following presentation on the next-operation recommendation, TIDY also stores targeted entities of operations when truly instantiating each goal (Line 17), composing a *seed entity set*, each of which is a seed entity, referring to the actual object following the clear user intention represented by this instantial goal. Note that, we only emphasize such entity's original attribute values (as ori_{entity} in Line 16) before meeting the goal (i.e., before applying all these operations), since TIDY in the following indeed tries to recommend next-operations upon entities that have not met such goal.

As a summary, TIDY now instantiates its selected parameterized goals into instantial goals, which suggest clear and executable user intentions for a sequence of related user operations. Then, based on such instantial goals, TIDY can make wise next-operation recommendation in the next step.

3.4. Stage 2: Next-operation recommendation

In this stage, TIDY tries to make a wise next-operation recommendation based on the obtained instantial goals and each goal's seed entities. Each of the former suggests a clear and executable user intention for an operation sequence, and each of the latter suggests a specific entity that actually has been manipulated in history for meeting a specific intention represented by its associated instantial goal. To do so, for each instantial goal, TIDY would first scan the document for identifying *relevant* entities to its seed entities, and treat them as possible candidates for the following operation recommendation with prioritization (step 1). Then, based on details of this instantial goal, TIDY generates concrete and executable next-operations accordingly for each related entity (step 2).

3.4.1. Step 1: Entity selection and prioritization

In this step, we introduce, given an instantial goal and its seed entities, how TIDY selects relevant entities from the whole document and prioritizes them for next-operation recommendation.

Given an instantial goal g_s and its seed entity set $ES_s = \{eS_1, eS_2, \ldots, eS_m\}$, TIDY first scans the document and obtain all remaining entities, i.e., $ES_r = \{eR_1, eR_2, \ldots, eR_n\}$. Then, for each entity eR_k in ES_r, TIDY calculates its relevance to ES_s, and based on it, prioritizes these entities. To be specific, TIDY calculates eR_k's relevance score to ES_s by accumulating all distances returned by comparing eR_k's value with those of entities in ES_s, with respect to each attribute in their attribute intersection, i.e., $I = attr(eR_k) \cap attr(ES_s)$, where $attr(x)$ returns an attribute set associated with an entity or an entity set x. Note that, if any eR_k's intersection set is empty, it would be removed from recommendation naturally. Details of eR_k and g_s's seed entity set ES_s relevance calculation are as follows (supposing g_s's targeted attribute set to be G).

$$relevance(g_s, eR_k, ES_s) = \frac{\sum_{a_i \in G} \frac{\sum_{a_j \in I} W(a_i,a_j) \times Dis(eR_k, ES_s, a_j)}{\sum_{a_j \in I} W(a_i,a_j)}}{count(G)}. \tag{73}$$

In this equation, $Dis(eR_k, ES_s, a_j)$ is designed to return a distance degree between eR_k and ES_s with respect to their corresponding values of attribute a_j in their attribute intersection I. To do so, TIDY treats numeric and tag attributes differently. When attribute a_j is a numeric attribute, $Dis(eR_k, ES_s, a_j)$ would compare whether the value of eR_k's attribute a_j is in the value range of ES_s's entities with respect to attribute a_j. If yes, it returns 1, or otherwise 0. When attribute a_j is a tag attribute, $Dis(eR_k, ES_s, a_j)$ compares whether the value of eR_k's attribute a_j occurs in any value of ES_s's entities with respect to attribute a_j. Formally, its calculation is as follows ($value()$ and $valueSet()$ return the corresponding value, respectively, for a certain entity or a set of values for a set of entities, with respect to a specific attribute):

$$Dis_{num} = \begin{cases} 1, & min(valueSet(ES_s, a_j)) \le value(eR_k, a_j) \\ & \quad \le max(valueSet(ES_s, a_j)), \\ 0, & otherwise. \end{cases} \tag{87}$$

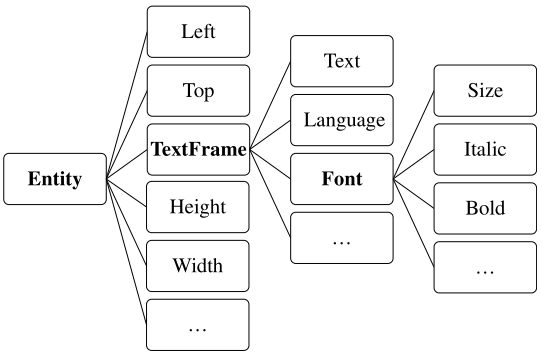

Fig. 3. Hierarchy tree structure of PowerPoint attributes.

$$Dis_{tag} = \begin{cases} 1, & value(eR_k, a_j) \in valueSet(ES_s, a_j), \\ 0, & otherwise. \end{cases}$$

With the distance degrees calculated for all attributes in I, TIDY also adopts different attribute weights $W(a_i, a_j)$ in calculating the final relevance score, since we believe that different attributes should naturally contribute differently to meet a certain instantial goal. In order to make such a weight allocation scalable and avoid complex tuning, TIDY adopts a domain hierarchy structure to measure the relevance of certain attribute a_j with respect to another attribute a_i obtained from goal g_s's interesting attributes. Such hierarchy structure can be accessible in many fields including rich-formatted documents, e.g., file system's standard hierarchy [12], HTML's keyword hierarchy structure [13], and etc. For example, for a popular rich-formatted document like PowerPoint, its attribute hierarchy is shown as in Fig. 3, with each attribute as a leaf node [14]. Therefore, it would be straightforward for TIDY to adopt the reciprocal of the path length (i.e., how many hops along this tree structure from one attribute to another) between two attribute nodes as weights, e.g., W("Left", "Size") = 1/4 (path: "Left" → "Entity" → "TextFrame" → "Font" → "Size"). More details about TIDY's application for PowerPoint would be given in Section 4.

As such, given an instantial goal and its seed entities, TIDY can produce relevance scores for all remaining entities, and then prioritize the entities in selection for next-operation recommendation. If necessary, one can also customize by pruning the selection using top N entities with highest.

3.4.2. Step 2: Next-operation generation and prioritization

In the last step, TIDY selects relevant entities to any specific instantial goal g_s based on their relevance scores calculated with respect to goal g_s's seed entity set ES_s. Then, TIDY can now generate the corresponding next-operation for recommendation by analyzing selected entities (e.g., e_{s1} and e_{s2}) and goal g_s.

For example, suppose $g_s = \langle (attribute_{width}), (value_{left} + value_{width} == 10) \rangle$ and the selected entities are $e_{s1} = \{attribute_{left} : 5, attribute_{width} : 4, \ldots\}$ and $e_{s2} = \{attribute_{left} : 3, attribute_{width} : 8, \ldots\}$. Then, the next-operations for recommendation would be: $op_{s1} = \langle e_{s1}, (attribute_{width} : 5) \rangle$ and $op_{s2} = \langle e_{s2}, (attribute_{width} : 7) \rangle$ in this case. Note that, such operation generation is straightforward by assigning suitable attribute values for goal's targeted attributes. If such value assignments have multiple solutions, TIDY returns a random one for application or multiple ones for its user to select from.

After generating next-operations for any obtained instantial goal and corresponding relevant entities, TIDY adopts a two-phase prioritization. First, for different instantial goals, TIDY prioritizes them by considering the entity number of their related seed entity sets, since we believe that with more entities in a goal's seed entity set, with more confidence it actually exposes the true user intention behind the given user operation history. Second, for a specific goal, TIDY also prioritizes its associated next-operations for any remaining entity in recommendation according to this remaining entity's relevance score with the goal's seed entities, as calculated in Section 3.4.1.

As a summary, combing these two stages together, TIDY can now identify user intentions from a given user operation history, and then make next-operation recommendation afterward. In the following, we continue to introduce how to apply TIDY to suggest and maintain the entity consistency in the PowerPoint application, and evaluate its effectiveness experimentally in turn.

Table 1
Selected attributes in applying TIDY to PowerPoint.

Selected attributes			
Id	Top	Left	Height
Width	Type	AutoShapeType	FontSize
FontName	FontItalic	FontBold	Underline
Text	ForeRGB	LineRGB	Weight
DashStyle	Rotation	EntryEffect	Visible
BackgroundStyle	BlackWhiteMode	HorizontalFlip	Child
LockAspectRatio	ShapeStyle	VerticalFlip	Style
3DVisible	3DDepth	AdvanceTime	Animate
ConnectionSiteCount	ChartColor	ChartStyle	ChartType
ObjectThemeColor	TintAndShade	FirstCol	FirstRow
HorizBanding	LastCol	LastRow	VertBanding
AutoSize	Transparency	AdvanceMode	ZOrderPosition
TopMargin	LeftMargin	BottomMargin	RightMargin

4. TIDY's application to PowerPoint

In this section, we introduce how to apply our TIDY approach to the popular rich-formatted document application PowerPoint for its automated entity consistency. We first give some details about TIDY's application on PowerPoint, especially concerning its goal library design and relevance calculation specific to the PowerPoint object hierarchy. Then, we exhibit on some realization details about our TIDY's prototype toolkit as a PowerPoint plug-in module.

4.1. Goal library design

In order to design a proper library for TIDY's application on PowerPoint, it is essential to clarify how to accordingly adapt TIDY's notations. For example, for PowerPoint, an entity usually refers to an instance of an object on a PowerPoint page, e.g., a rectangle, a circle, a textbox, a picture, a chart, etc. To better describe and distinguish entities and their associated user operations, PowerPoint also provides massive attributes in its object model [14]. To apply TIDY, we select 52 representative attributes in PowerPoint's normal usage, and details are shown in Table 1. These selected attributes concern diverse entity characteristics like object location, coloring, text font, size, style, etc, and those selected attributes and corresponding values would be used to identify entities and entity operations when applying TIDY.

After that, based on such attributes, we accordingly design a goal library including 26 different parameterized goals, as shown in Table 2, which achieve diverse and representative layout consistency and style consistency for PowerPoint. For example, alignment is one of the most representative layout consistency types for PowerPoint entities, and in designing such a goal library, we consider multiple alternatives of alignment for completeness. As such, the goal library would then be used in TIDY and support its user intention identification, and those parameterized goals in the library would be instantiated during TIDY's analyses to achieve the next-operation recommendation.

Table 2
TIDY's goal library designed specific for PowerPoint.

Type	Targetd attribute	Expressional descriptions	General description
1	Left	$value_{Left} == x$	Left-aligned
	Top	$value_{Top} == x$	top-aligned
	Width	$value_{Left} + value_{Width} == x$	Right-aligned (change Width)
	Left	$value_{Left} + value_{Width} == x$	Right-aligned (change Left)
	Height	$value_{Top} + value_{Height} == x$	Bottom-aligned (change Height)
	Top	$value_{Top} + value_{Height} == x$	Bottom-aligned (change Top)
2	FontSize	$value_{FontSize} == x$	Unify the font size
	FontBold	$value_{FontBold} == x$	Unify whether bold or not
	FontItalic	$value_{FontItalic} == x$	Unify italic or not
	Underline	$value_{Underline} == x$	Unify underlined or not
	FontName	$value_{FontName} == x$	Unify the font (e.g., Arial)
	ForeRGB	$value_{ForeRGB} == x$	Unify the shape-filling color
	LineRGB	$value_{LineRGB} == x$	Unify the shape outline color
	Weight	$value_{Weight} == x$	Unify the shape outline weight
	DashStyle	$value_{DashStyle} == x$	Unify the outline dash style
	EntryEffect	$value_{EntryEffect} == x$	Unify the entry effect
	Rotation	$value_{Rotation} == x$	Unify the rotation angle
	Width	$value_{Width} == x$	Unify the width
	Height	$value_{Height} == x$	Unify the height
3	Left	$\Delta value_{Left} == x$	Move by the same distance horizontally
	Top	$\Delta value_{Top} == x$	Move by the same distance vertically
	Width	$\Delta value_{Width} == x$	Unify the variation of width
	Height	$\Delta value_{Height} == x$	Unify the variation of height
	FontSize	$\Delta value_{FontSize} == x$	Increase or decrease the same font size
	Weight	$\Delta value_{Weight} == x$	Unify the variation of outline weight
	Rotation	$\Delta value_{Rotation} == x$	Rotate by the same angle

Type: 1. layout consistency 2. style consistency 3. both layout and style consistency.

4.2. Powerpoint object hierarchy

As we mentioned in Section 3.4.1, when calculating relevance scores for selecting entities for recommendation, TIDY adopts a domain hierarchy structure to measure attribute weights $W(a_i, a_j)$. When applying TIDY to PowerPoint, we customize this application by using a general PowerPoint's attribute hierarchy as shown in Fig. 3, with each attribute as a leaf node [14]. We defines $hops(a_i, a_j)$ to calculate how many hops along this tree structure between two attribute nodes a_i and a_j, and $W(a_i, a_j)$ to be its reciprocal. Then, the corresponding $W(a_i, a_j)$ is calculated as follows:

$$W(a_i, a_j) = \begin{cases} 1/hops(a_i, a_j), & i \neq j, \\ 1, & i = j. \end{cases}$$

For example, W("Left", "Size") is 1/4, since $hops$("Left", "Size") = 4 (path "Left" → "Entity" → "TextFrame" → "Font" → "Size" has four hops).

4.3. Prototype toolkit details

We implemented TIDY as a VSTO (Visual Studio Tools for Office) plug-in module for being integrated into PowerPoint using C#. The plug-in's user interface is shown as Fig. 4.

A user can turn on and off TIDY's recommendation service by simply clicking the start and stop buttons on the right column of the screen. During TIDY's execution, it silently collects the user's operations on PowerPoint's editing page in the middle. If TIDY identifies a clear user intention, it will provide next-operation recommendation with several alternatives, and each recommended next-operation is shown as a preview of the operation effect and a small blue button at the top right-hand corner. One can choose to click the blue button to accept a corresponding recommended operation so as to directly apply its effect to the middle edit page. Since PowerPoint itself does not support buttons on the edit page, we used the mouse hook technique [15] to visually present our recommendation as a button-alike effect to the user as illustrated. To do so, all previews and buttons for recommendation are created temporarily as PowerPoint shapes, and the mouse hook returns the clicked button. If the user does not click any button, all previews and buttons for recommendation will automatically disappear once the next user editing operation occurs. Note that, since the position of the edit page is not always fixed, the plug-in also adopts automatic position correction to ensure TIDY to work as expected.

Besides, to allow users to undo and redo their choices on recommended operations occasionally, we also implemented additional undo and redo functionalities in this plug-in service, since sometimes users may accidentally accept/skip recommendations unexpectedly.

Furthermore, TIDY targets at maintaining entity consistency for rich-formatted documents, by recommending operations to achieve consistent entities in practice. For those user intentions that are not relevant to entity consistency (e.g., a user changes a rectangle's alignment, and then changes another rectangle's color to yellow and a circle's color to yellow too), it would not what TIDY focuses on. Enforcing TIDY to work in this situation would clearly produce recommendations unexpectedly. We consider this to belong to users' choices. To avoid causing confusion or disturbance to users when they are not for entity consistency, we have designed a stop button in the TIDY plug-in module, for users to temporarily disabling TIDY's recommending service.

5. Evaluation

In this section, we evaluate the performance of TIDY, concerning its application to the popular rich-formatted document application PowerPoint for its automated maintenance of entity consistency.

5.1. Research questions

We aim to answer the following three research questions:

RQ1 (Effectiveness): *How effective is* TIDY *on its wise next-operation recommendation for achieving PowerPoint's entity consistency?*

RQ2 (Factors): *How do different factors affect* TIDY*'s effectiveness?*

RQ3 (Overhead): *How much overhead does* TIDY *take to make recommendations, and does it compromise* TIDY*'s effectiveness?*

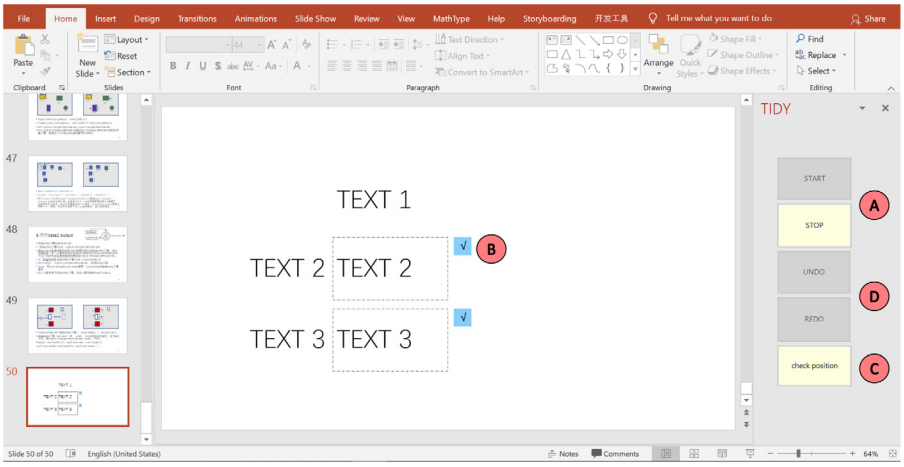

Fig. 4. (A) Start and stop buttons. (B) After the first textbox (TEXT 1) was moved to right, TIDY gave recommendations (two dotted boxes as the preview) for moving the second (TEXT 2) and third (TEXT 3) textboxes to right and being aligned with the first textbox. (C) Position correction for the mouse hook. (D) Undo and redo buttons. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

5.2. Experimental design and setup

To answer the three research questions, we first introduce how to obtain practical user operation history, and its true next-operations would serve as the ground truths for our experiments. Then, we explain the experimental setup and how to answer the three research questions individually.

5.2.1. Experimental preparation

To obtain practical user operations and their corresponding next-operations on PowerPoint, we invited 21 participants for 5 editing tasks concerning diverse and popular PowerPoint functionalities.

Participants and tasks. On one hand, we invited 21 participants with diverse PowerPoint skills, including 3 teachers, 6 Ph.D. students, 9 MSc students, and 3 undergraduate students. In order to avoid biases, we make sure all participants to be unaware of our TIDY methodology. On the other hand, adapted from popular examples on PowerPoint's online forums [16], we designed 5 tasks including a total of 21 mini-tasks for each participant. Those 5 tasks contain diverse categories of entities in PowerPoint (e.g., shapes, textboxes, art words, pictures, etc.), and the task design covers diverse user intentions upon each task. Details are shown in Table 3.

Collection process. We asked all participants to finish such tasks alone, and collected their user operations accordingly. To collect the operation sequences generated by participants non-intrusively, we additionally developed a plug-in for logging operations in PowerPoint. It would silently log participants' operations upon each page, and accordingly store them to the disk. In our experiments, we sent the tasks and the plug-in to all participants, and asked them to complete all the tasks with the log plug-in enabled. Meanwhile, we also asked them to use screen-capturing software to record the entire process of completing the tasks, for double-checking.

Ground truths. As a total, we collected 101 valid user operation history logs (manually removing 4 invalid ones when participants failed to finish tasks), each of which refers to an operation sequence representing a participant's editing history for one task. On average, each participant spent 20 min finishing these tasks. To obtain our expected ground truths of an operation history and corresponding next-operation, we partition each log to be a prefix subsequence representing an operation history (namely *history sequence*) and its following subsequence to be its next-operation sequence (namely *follow-up sequence*). That is, for a log with length n, i.e., $op_1, op_2, \ldots, op_n$, one can easily partition it by x, i.e., any value in $[1, n)$. Then, it regards the prefix subsequence $op_1, op_2, \ldots, op_x$ as its history sequence, and the next operation of that subsequence $op_{x+1}, \ldots, op_n$ as its follow-up sequence for next-operations. Such instances of both history sequences and corresponding follow-up sequences constitute natural datasets for evaluating TIDY, with ground truths available. For example, in order to complete the three mini-tasks of task 1, participant #19 performed a total of 25 operations, and therefore we naturally obtained 24 instances from this log.

Generally, each log is an operation sequence performed by a participant to complete our tasks as shown in Table 3, and each task is composed of several mini-tasks. For the integrity of ground truths, if a participant does not complete all mini-tasks in a task, we consider the corresponding log to be invalid for experiments (e.g., potentially biased to experimental results). We obtained a total of 2363 instances by partitioning 101 valid user operation history logs, after discarding those logs indicating that participants failed to finish any mini-task. The average length of history sequences and that of follow-up sequences are 17.2 and 17.1, respectively. To be specific, we removed three task logs from participants #4 (task 1), #12 (task 1), and #13 (task 2), since they all accidentally skipped one of their mini-tasks. Moreover, we did not collect any task log from participant #15 (task 5), since he skipped the entire task 5 accidentally in our collection. Note that such 2363 instances in collection cover all 26 goals in TIDY's goal library, as shown in Table 2. These instances would be used for evaluating TIDY in the following.

5.2.2. Experimental setup

To evaluate TIDY and answer the aforementioned three research questions, we designed the following independent variables:

- *Hit distance limit*. We restrict the hit distance limit of TIDY's recommended next-operations in the follow-up sequence in evaluation. We regard "hit" to be satisfied, only when TIDY's recommended next-operation appears within the hit distance limit in its corresponding follow-up sequence. We controlled the hit distance limit from one to fifteen with a pace of one since we consider the operations too far to be less relevant in recommendation. We set the default value of the hit distance limit to one, for the reason that it denotes the most challenging setting. That is, for the given history sequence $op_1, \ldots, op_x$, when setting the default value for the hit distance limit to be one, "hit" is restricted to be satisfied only when TIDY successfully recommends the exact next user operation op_{x+1} without any mistake. Therefore, this setting represents the most restrict application scenario, which is also the most expected by users in practice.

Table 3
Designed tasks for all participants.

Task description
Task 1: Arrange tables and shapes for alignment
1. Move all tables to make them left-aligned in any arbitrary location. 2. Arrange all rectangle shapes to three horizontal rows by color and make rectangles in the same row bottom-aligned (overlapping between shapes is always allowed). 3. Pick a rectangle shape in an arbitrary color, change its length without moving it, and then make the lengths of other rectangles in that color to the same as it.
Task 2: Adjust charts and art words for consistent styles
1. Some charts or art words do not have animation effects upon entering this page. Please add an arbitrary entrance animation effect for each of them. 2. Pick all charts or all art words (choose only one type), and turn their entrance animation effects into the same one. 3. Adjust the lengths of art words and make them right-aligned in an arbitrary place. 4. Pick an arbitrary color, and change the borders of all art words or charts (choose only one type) to that color.
Task 3: Paint and adjust shapes for consistent styles
1. Draw four or five rectangles of different sizes and four or five circles of different sizes. 2. Pick four or more tilted images or textboxes, and rotate them to upright. 3. Adjust the sizes of all shapes of an arbitrary type (rectangle or circle) to be consistent. 4. Pick an arbitrary color, and fill all shapes of an arbitrary type with this color.
Task 4: Adjust textboxes for consistent styles
1. Pick any used font color, and change textboxes' font in this color to Times New Roman. 2. Pick any used font color, and change all borders of textboxes in this color to the same weight, such like 1.5, 2.25, or 3 pt. 3. Pick a state between tilting or not tilting, and change all font sizes of the textboxes with this state to 16. Then change font sizes of textboxes with the other state by the same reduction. 4. Change all bold textboxes or all non-bold textboxes to italics. Then add underline style to the textboxes of the other type. 5. Pick three arbitrary non-bold textboxes and make their text bold.
Task 5: Paint and adjust shapes for alignment and consistent styles
1. Insert x circles, y rectangles, and z triangles to make $x + y + z = 15$. 2. Move all shapes of one type to make them top-aligned in any arbitrary place. 3. Take all shapes of one type and increase their borders by the same weight. 4. Take all shapes of one type and change their line types of borders into short dash. 5. Take all shapes of one type and rotate them by 90 degrees clockwise.

- *Slot limit*. Our TIDY approach makes its recommendation with prioritization. In order to better evaluate how effective its prioritized recommendation is, we control this variable to denote the number of recommendation candidates that TIDY can recommend for each specific user intention. We control it from one to twenty with a pace of one. We set the default value of it to be 10 since its corresponding TIDY's effectiveness can be stably satisfactory, as investigated in Section 5.3.2 later.
- *Tolerance size*. As mentioned in Section 3.3.1, TIDY is integrated with a special treatment for tolerating unrelated operations during examining practical sequences. We use tolerance size to represent the specific budget, which refers to the specific budget for consecutive matching failures allowed in TIDY's goal examination every time. When the tolerance size is restricted to k, TIDY would conduct its backward analysis by allowing k unrelated operations continuously in one goal examination. That is, as long as the number of continuous unrelated operations during matching is not greater than k, the backward analysis will continue.
- *Goal prioritization*. Our TIDY approach also identifies different user intentions as its instantial goals with prioritization in analysis. In order to better evaluate how effective its identified user intentions' prioritization is, we also control this variable to further investigate how differently TIDY's prioritized user intention affect its performance, thus suggesting the effectiveness of TIDY's user intention prioritization.
- *Task*. We control this variable to study how each individual task setting affects TIDY's performance.
- *Participant*. We control this variable to study how different participant characteristics affects TIDY's performance, e.g., participant occupation, gender, etc.

Then, to evaluate TIDY's performance concerning its effectiveness and efficiency, we designed the following dependent variables:

- *Hit rate*. For any given instance, if TIDY can successfully recommend at least one operation which is in the follow-up sequence within the controlled hit distance limit, we regard it as "hit". Then, accordingly, we measure the hit rate as the proportion of "hit" times against 2363 instances.
- *Time overhead*. We measure the time overhead by the amount of time TIDY spent on making recommendations for any given history sequence.

All experiments were conducted on a commodity PC with Intel(R) Core(TM) i5-9500 CPU @ 3.00 GHz with 16.0 GB RAM. The machine was installed with MS Windows 10 and Oracle Java 15.

To answer RQ1 (Effectiveness). We calculate the hit rate of TIDY's next-operation recommendations for all collected history sequences concerning its follow-up sequences as ground truths. We control the hit distance limit to 1, slot limit to 10, and tolerance size to 1, respectively, as a default setting when answering this research question.

To answer RQ2 (Factors). We first assign different values to investigate how different hit distance limits, slot limits, and tolerance sizes affect TIDY's effectiveness especially concerning hit rates. Meanwhile, we also look into TIDY's stableness and variance in its effectiveness with respect to other factors, e.g., goal prioritization, tasks, and participants.

To answer RQ3 (Overhead). We measure TIDY's time overhead in making recommendations to all history sequences, and investigate whether the overhead would compromise TIDY's effectiveness.

5.3. Experimental results and analyses

We report and analyze experimental results, and answer the preceding three research questions in turn.

5.3.1. RQ1: Effectiveness

To evaluate TIDY's effectiveness, we conducted TIDY's recommendation for all collected instances, each of which include a history sequence as the fed data to TIDY and a follow-up sequence as the ground truth for evaluation. As shown in Table 4, we collected 2363 instances in total, which concern all 21 participants and our designed five tasks, as mentioned earlier in Section 5.2.1.

Table 4
Descriptions and hit rates analyses for all participants and tasks (UG: undergraduate; ins.: instances).

Id	Description	# ins./# tasks	Hit rate (task 1, 2, 3, 4, 5)
1	MSc, female	125/5	76.0% (66.7%, 63.6%, 72.2%, 82.4%, 83.3%)
2	PhD, female	114/5	80.7% (77.8%, 77.8%, 75.0%, 83.0%, 84.2%)
3	PhD, male	109/5	81.7% (80.8%, 84.6%, 77.8%, 81.8%, 82.4%)
4	MSc, male	90/4	75.6% (–, 66.7%, 63.6%, 77.6%, 83.3%)
5	MSc, male	114/5	76.3% (62.5%, 85.7%, 81.8%, 77.8%, 83.3%)
6	MSc, male	120/5	70.8% (57.6%, 75.0%, 70.0%, 73.1%, 88.2%)
7	PhD, male	113/5	83.2% (83.3%, 83.3%, 77.8%, 85.0%, 83.3%)
8	Teacher, male	118/5	82.2% (85.7%, 70.0%, 75.0%, 83.7%, 84.2%)
9	PhD, male	100/5	71.0% (79.2%, 66.7%, 72.7%, 61.9%, 82.4%)
10	Teacher, male	113/5	71.7% (65.5%, 87.5%, 75.0%, 71.4%, 73.7%)
11	PhD, male	118/5	79.7% (82.1%, 87.5%, 84.6%, 74.0%, 84.2%)
12	Teacher, male	84/4	76.2% (–, 75.0%, 66.7%, 76.0%, 83.3%)
13	MSc, male	102/4	76.5% (83.3%, –, 80.0%, 74.0%, 72.2%)
14	MSc, male	123/5	80.5% (85.2%, 83.3%, 68.8%, 80.0%, 83.3%)
15	MSc, male	103/4	73.8% (74.1%, 84.6%, 76.9%, 70.0%, –)
16	MSc, female	129/5	67.4% (68.6%, 66.7%, 66.7%, 67.3%, 66.7%)
17	UG, female	116/5	81.0% (80.6%, 77.8%, 66.7%, 84.3%, 81.3%)
18	UG, male	113/5	80.5% (84.0%, 63.6%, 83.3%, 81.6%, 81.3%)
19	UG, male	109/5	83.5% (83.3%, 85.7%, 66.7%, 85.4%, 85.7%)
20	MSc, female	133/5	75.2% (78.6%, 81.8%, 72.7%, 68.6%, 83.3%)
21	PhD, male	117/5	79.5% (71.4%, 66.7%, 75.0%, 86.0%, 83.3%)
In total:		2363/101	77.3% (76.0%, 76.7%, 73.7%, 77.4%, 81.7%)

As such, upon the remaining 2363 instances for experiments, we set with the default hit distance limit of one (i.e., only TIDY recommending the exact op_{x+1} achieves "hit"), slot limit of 10, and tolerance size of one as mentioned before in Section 5.2.2, TIDY's hit rate for all participants is 77.3% on average (range: 67.4–83.5%), concerning averaged hit rates of 76.0% (range: 57.6–85.7%), 76.7% (range: 63.6–87.5%), 73.7% (range: 63.6–84.6%), 77.4% (range: 61.9–86.0%), and 81.7% (range: 66.7–88.2%) for each task, respectively.

As a summary, in answering RQ1, our experimental results suggest TIDY's promising effectiveness on its wise next-operation recommendation by its default setting, and its effectiveness generally holds for different concerned tasks and participants with little variance.

5.3.2. RQ2: Factors

We next study how different settings for the hit distance limit, slot limit and tolerance sizes affect TIDY's effectiveness, and how its effectiveness varies with different prioritized goals, tasks and participants.

Hit distance limit. As mentioned earlier, a hit distance limit is designed for restricting how far TIDY's recommended operations appearing in a follow-up sequence can be still regard as "hit". Let a history sequence be $op_1, op_2, \ldots, op_x$, and its follow-up sequence be $op_{x+1}, op_{x+2}, \ldots, op_n$. When a hit distance limit is set to k, then only TIDY's recommended operations matching any operation between $op_{x+1}, \ldots, op_{x+k}$ can be regard as "hit". We next investigate how different settings for the hit distance limit affect TIDY's effectiveness concerning its hit rate.

Fig. 5 shows how TIDY's hit rate changes with the increasing hit distance limit. Note that, to avoid the confusion and bias, when calculating the corresponding hit rates for a certain hit distance limit k, we only take those instances whose follow-up sequences have no less than k operations. That is, when calculating hit rates for different hit distance limits, the figures refer to different sets of instances. This explains why the hit rate might decreases in Fig. 5 when the hit distance limit increases. From the figure, we can observe that, with the increase of the distance limit, there is an obvious increase at first (until around eleven), and then becomes steady gradually. This suggests: (1) with the distance limit relaxed a little bit, hit rates can be increased to some extent since more cases are likely to be regarded as "hit", and (2) such increasing trend no longer continues when the distance limit reaches a certain value, since a user intention is typically not associated with too many user operations in a large scope and its increasing trend will converge when the distance limit already reaches the end of the user intention's associated operations.

Slot limit. As mentioned earlier, the slot limit is designed for controlling how many next-operations TIDY can recommend for each specific goal. Since TIDY recommends its next-operations with prioritization, we choose to control different slot limits to further evaluate how its prioritization treatment contributes to its recommendation and how effective its recommended next-operations with the highest prioritization are. When the slot limit is set to be k, TIDY is restricted to have only k options for next-operation recommendation for each analyzed user intention, and therefore we can accordingly calculate the corresponding hit rate, i.e., top-k hit rate.

Fig. 6 shows the hit rates for different slot limits. From the figure, we can observe that: (1) with an increasing slot limit, TIDY achieves a higher hit rate, which increases rapidly at first and becomes steady quickly; (2) TIDY's effectiveness holds for different slot limits with the top-3 hit rate being 60.8%, top-5 hit rate being 73.2%, and top-10 hit rate being 77.3%; (3) hit rates become relatively steady when k is more than ten, suggesting TIDY's top ten prioritized next-operations recommendation are already most effective in practice; (4) although TIDY's top-1 hit rate is relatively low, it can increase largely when the slot limit is relaxed a little bit to two or three. As such, those observations can promisingly suggest the advantages and effectiveness of TIDY's prioritization treatment on recommended next-operations. When further balancing different slot limit settings, their corresponding difficulties for users to choose recommendations, and finally achieved hit rates, we suggest a suitable slot limit to be 10, since it is still relatively easy for users to quickly scan and choose recommendations and also brings acceptable effectiveness (hit rate of 77.3%). Still, to keep our investigation in the experiments complete, we preserve its maximum value of 20 when investigating this factor's impact on TIDY's effectiveness.

Tolerance size. Fig. 7 shows how different tolerance sizes would affect TIDY's effectiveness. We investigate hit rates of instances when controlling different tolerance sizes in implementing TIDY. From the figure, we can obverse that: (1) TIDY's hit rates can be slightly increased by such tolerating treatments, showing that such tolerating treatment can indeed alleviate the problem of unrelated operations occurring unexpectedly at the middle of practical sequences; (2) TIDY's hit rates reach to the highest when the tolerance size is set to 2, and the rates slightly decrease after that. Note that we observe that such improvement exists but is not large in our experiments. This could attribute to two facts: (1) the collected sequences for experiments were not that noisy, (2) the tolerance treatment can help alleviate on this issue.

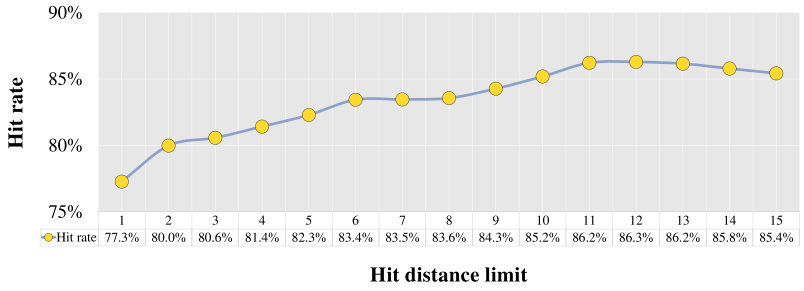

Fig. 5. Averaged hit rates with different hit distance limits.

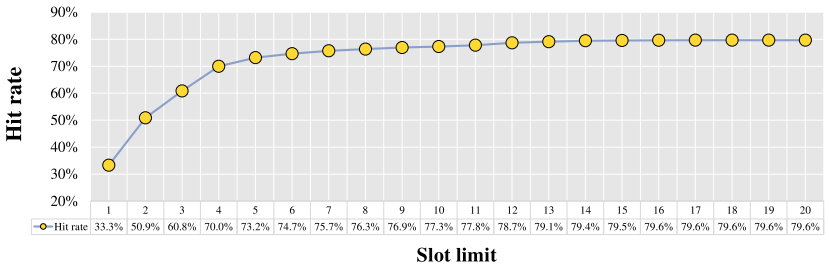

Fig. 6. Averaged hit rates with different slot limits.

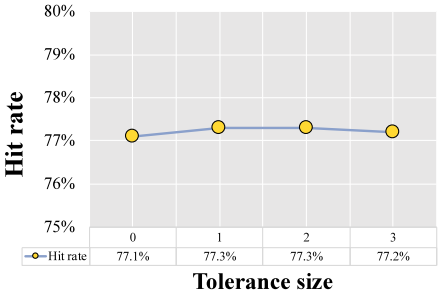

Fig. 7. Averaged hit rates with different tolerance sizes in TIDY.

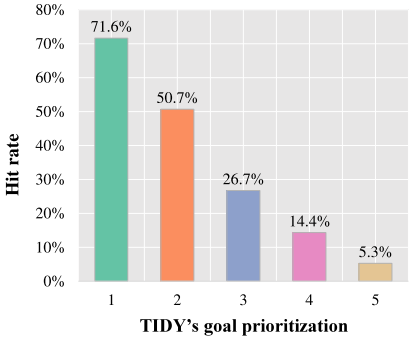

Fig. 8. Averaged hit rates on the top fifth prioritized goals.

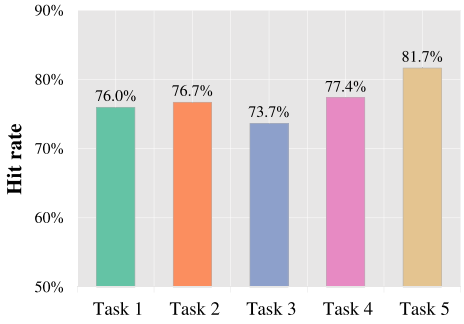

Fig. 9. Averaged hit rates of instances from different tasks.

Goal prioritization. During TIDY's analysis upon a given history sequence, it may identify different user intentions with prioritization. To study whether TIDY's prioritization of different identified user intentions is reasonable, we individually calculated corresponding hit rates on the identified user intentions with a prioritization ranking from one to five. Results are shown in Fig. 8. We can observe that, TIDY can effectively identify the user intention with highest hit rates (71.6%) as the highest prioritized intention, and when TIDY's identified goal prioritization decreases, the corresponding hit rate decreases as expected. This not only suggests the effectiveness of TIDY's identified user intentions, but also indicates that the rational of its goal prioritization treatment is reasonable and effective. Moreover, since the hit rate of TIDY's top-1 user intentions already reaches a satisfactory value of 71.6%, this also suggests an alternative for TIDY to prune unnecessary user intentions during its analyses without affecting its overall effectiveness.

Task. To investigate how different task designs may affect TIDY's effectiveness, we look into hit rates of instances from each individual task. Fig. 9 shows hit rates results of instances from different tasks. We can observe that: (1) for all the five tasks, TIDY's hit rates are consistently promising (ranging from 73.7–81.7%); (2) hit rates for different tasks are generally stable, with small variance less than 10%. This suggests TIDY's general and stable effectiveness across different tasks.

Participant. To further investigate how different participant characteristics may affect TIDY's effectiveness, we investigate hit rates of instances from each different category of participant characteristics like occupation and gender. Fig. 10 show hit rates results of instances from participants with different occupations and genders, respectively. More detailed hit rates for individual participant can be found in Table 4.

We can observe that TIDY's hit rates of instances from participants individually (Table 4), or participants with certain occupations and genders (Fig. 10) are generally stable, with only slight differences. By further examining each participant's recorded screen videos during completing tasks, we observe that such slight differences on hit rates

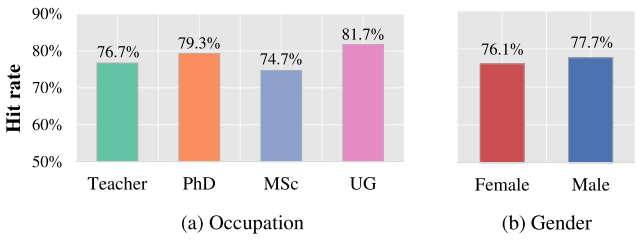

Fig. 10. Averaged hit rates for participants with different occupations and genders.

Table 5
TIDY's averaged time overhead per instance.

Task	Averaged time overhead (ms)
1	16.2
2	2.7
3	8.0
4	4.1
5	5.0
Total :	7.2

might be due to participants' different editing habits. For example, when completing task 1 (i.e., arranging all rectangle shapes to three horizontal rows by color and make rectangles in the same row bottom-aligned), participant #6 first roughly arranged the rectangles in three rows according to their colors, and then carefully aligned them together. On the other hand, participant #16 added some unnecessary operations to make shapes to be aligned vertically, even though the task does not require that. Such habit differences among participants might somehow explain the variance among their corresponding hit rates in Table 4. Moreover, frankly speaking, we actually observed that undergraduate students were more willing to complete tasks according to instructions step by step, while other participants tended to scan tasks at first and then edit without following instructions rigorously. This may explain why "UG" participants achieves the highest hit rate in Fig. 10. Generally, different occupations and genders had bought only marginal variance to TIDY's effectiveness, and this also suggests TIDY's stable effectiveness across different participants.

As a summary, by investigating into different factors that might affect TIDY's effectiveness, we observe that TIDY's effectiveness consistently holds concerning different factors. When the hit distance limit and slot limit increase, its effectiveness of hit rates increases accordingly at first and quickly becomes stable with little variance.

5.3.3. RQ3: Overhead

We measured the time overhead when applying TIDY to making next-operation recommendations for all collected 2363 instances and calculated the averaged time overhead per instance for different tasks.

As shown in Table 5, TIDY took only 7.2 ms on average to make recommendations for per instance, suggesting its great efficiency (overhead negligible). Besides, we do observe that TIDY's time overheads varies among different tasks. For example, in Table 5, instances of task 1 took the largest averaged time overhead, i.e., 16.2 ms, larger than those of the other four tasks' instances on average, i.e., 2.7–8.0 ms. We looked into its underlying reason, and figured out that this is mainly because task 1 actually involves plenty of user operations concerning location-related attributes (e.g., i.e., Left and Top), which are associated with much more parameterized goals in the designed goal library in Table 2, compared to other attributes, e.g., coloring attributes like EntryEffect or LineRGB corresponding to only one parameterized goal. This naturally brings different complexities to TIDY's calculation and analysis, and explains its relatively larger time overhead on task 1. Even so, we still consider TIDY's time overhead marginal by costing only tens of milliseconds per instance.

As a summary, we believe that TIDY is very efficient by only costing time overheads from several to several tens of milliseconds per instance, and such overhead is clearly acceptable at runtime, not compromising its promising effectiveness observed in answering RQ1 and RQ2.

5.4. Threats to validity

One may concern that the evaluation in TIDY's application to PowerPoint may not be generalized to other rich-formatted document applications. We alleviate this threat as follows. First, our TIDY approach is proposed and designed independently of PowerPoint. All concepts inside the approach, such as entities, operations, and goals, can be easily generalized to other rich-formatted document applications. The only domain-specific structure is the hierarchy structure used in calculation, and it can be replaced by other accessible structures in many fields including rich-formatted documents. Second, our selected application PowerPoint is one of the most popular and commonly used rich-formatted applications. We believe that applying TIDY to and evaluating it on PowerPoint is representative.

Besides, one may also concern that our invited participants and designed tasks may not be representative of common PowerPoint users and tasks. In our participant selection, we tried our best to invite more participant types with diverse characteristics such as occupation, and gender. Moreover, in order to avoid experimental biases, we also make sure all participants to be unaware of our TIDY approach and they were restricted to complete all tasks individually, without any influence from other researchers or participants. In our task design, we designed our tasks by adapting popular examples on PowerPoint's online forums [16], which cover diverse PowerPoint popular functionalities and objects in PowerPoint (e.g., shapes, textboxes, art words, pictures, etc.). Also, our task design allows some degree of freedom for participants in order to better investigate TIDY's general effectiveness to unexpected situations.

6. Discussion

In this section, we discuss some issues regarding TIDY's usage in handling challenging sequences in practical scenarios, and its generalization ability to more rich-formatted documents.

6.1. Handling challenging sequences in practical scenarios

Regarding challenging sequences in practice, we emphasize on three cases: (1) sequences with only extremely few operations, (2) sequences with disturbance by unexpected operations in the middle, (3) sequences with complex intentions.

Considering that users may only provide extremely few examples in history, this brings challenges to not only TIDY, but also all work for operation recommendations. Still, we believe that with the aid of its domain-specific goal library for modeling common user intentions, TIDY can restrict the search space, and make its recommendations effectively even with only few examples in practice. To further alleviate possible concerns about this case, we additionally provide some experimental data here for a better explanation. In our experiments, when we partitioned instances according to the number of related operations under consideration concerning each recommendation (i.e., the number of seed entities for the highest prioritized intentions), the corresponding hit rates for TIDY's effectiveness are 63.3%, 84.6%, 84.6%, and 79.0% when using only 1, 2, 3, and 4 operations, respectively. We observe that when only one example is provided, TIDY's effective on hit rate is 63.3%, seemingly not high but actually already surprisingly high. The reason is that only one example is somewhat a disaster for any PBE work that aims to make a useful recommendation (one example can be interpreted in any way). TIDY's effectiveness for these few numbers of examples should attribute to it goal library modeling, as

we mentioned earlier. Still, no one can argue that TIDY is able to cope with any scenario where only one example is available. We can observe that TIDY's effectiveness can quickly increase to a satisfactory degree (around 80% hit rate) when it takes two or three examples. We consider it practically useful for TIDY's target scenarios.

Considering that users may possibly not stick to one intention due to unexpected disturbance (e.g., jumping to some irrelevant actions unexpectedly), some unexpected operations may occur in the middle of a collected sequence in practice and somehow make a goals associated operations non-consecutive, although we believe that noisy operations tend to be few in a sequence since users typically tend to complete one task before moving to another, in order to be efficient and free of disturbance. Still to alleviate possible problems, we adopted a tolerance treatment in TIDY to allow few unexpected operations occurring (as discussed in Section 3.3.1). Our experiments also show that such tolerance can somewhat contribute to TIDY's effectiveness.

Moreover, some practical sequences may even arise some complex intentions which involve several operations rather than only one for an entity each time. For example, a user may conduct several actions as a whole package to different entities one by one, i.e., for each entity, first changing its font color to red and then changing its font size to 14. This intention is complex and beyond our original goal library, since it combines several concrete operations, and its goal examination may differ from TIDY's original treatment. Currently, TIDY might not support this. Yet we consider that it could be supported by extending TIDY's goal library with such "combined intentions" and associating a goal with different combining patterns for next-operation recommendations. We leave it to our future work. Note that, even if TIDY does not support combined intentions currently, TIDY is still capable of handling many regular entity consistency tasks.

6.2. Generalization to more rich-formatted documents

TIDY is proposed to maintain the entity consistency for rich -formatted documents. Generally, applying TIDY to a specific type of rich-formatted documents is as follows: modeling the concerned attributes (could be diverse), designing a suitable goal library (could cover common user intentions), and then based on these preparations, using TIDY accordingly. In this article, we implemented TIDY as a plug-in module into PowerPoint, a commonly used rich-formatted document application. Based on it, one could enrich TIDY's application upon PowerPoint, e.g., considering new intentions like "unifying entities' shadow effects". Such application is straightforward, as long as shadow-related attributes are modeled and used in the goal library.

One could also consider applying TIDY to other types of rich-formatted documents from scratch. We give a brief example for guidance. For similar presentation-based applications like Prezi, applying TIDY to them is similar to that to PowerPoint. For other rich-formatted documents that might be some-what different from PowerPoint, e.g., online drawing sites (which draw diagrams like UML diagrams and data flow diagrams) or mind mapping modules (embedded in some software applications), TIDY can still be used by a few adaptations. Generally, TIDY's key concepts, like "entity", "operation", and "goal", should be mapped to new elements. For example, in a diagram-drawing application, one should extract its elements (e.g., nodes in a mind map, classes in class diagrams, and files in DFD) as TIDY's entities with their associated attributes, and model possible actions relating to these elements as TIDY's operations (e.g., drawing graphs, lines, or changing coloring, text font, text size, and so on). Then, one proceeds to build TIDY's goals upon these elements, attributes, and actions by digging into users' popular intentions in practice. For example, in a UML diagram-drawing application, users tend to draw an implementation class diagram when there is an interface diagram. This intention can be modeled as a goal of maintaining the entity consistency between interface and implementation class diagrams. Based on such goals, TIDY can then recommend related next-operations like drawing implementation class diagrams for an interface with no implementation class, or recommend method options for existing implementation classes by analyzing its corresponding interface diagram.

7. Related work

This work aims to maintain entity consistency and relates the most to existing research on PBE work. As a sub-field of program synthesis, PBE aims to synthesize an intended program based on given examples, which are supposed to be representative. It is quite unlike traditional program synthesis, which requires specifications usually described by logical formulas [17–19]. PBE techniques have been widely proposed for applications in many fields [20–22], e.g., repeating structured drawing [23], remodeling [24,25], spreadsheets [4,8], file management [9], and data parsing and extraction [3,26]. There could be two main lines in the PBE field, i.e., data transformation and code transformation.

Data transformation. This line of PBE work aims to automatically transform data from its original format into another format that can be better analyzed and visualized. There are challenges because the analyzed data can be restricted by various types of documents, such as text files, PDF documents, HTML documents, and spreadsheets. Without PBE work, it was estimated that data scientists have to spend 80% of their time on doing data transformation [27], while PBE can effectively help to conduct data transformation [28–30]. For example, a built-in feature in MS Excel, FlashFill [4,7], is a typical PBE-based tool for automatically performing string conversions for cells in Excel files. By using provided examples of input–output string cells, it automatically generates programs to perform string conversions as expected. FlashExtract [3], delivered in Windows 10 as convertFrom-String cmdlets, is also another popular PBE-based tool which can extract data from semi-structured documents, such as HTML documents and text files. It can successfully produce programs to collect all samples of a field in the output data schema with negative/positive instances of that field provided by users.

Code transformation. This line of PBE work aims to perform automatic code conversions, since it is observed that around 40% of developers' energy was spent on executing repeated modifications to the application code, which is resource-wasting. PBE can be a great assistance to this field [31,32], to improve the performance of the code conversions. For example, as the internal Linux libraries continue to evolve, Andersen et al. [33,34] proposed a method to help Linux device drivers update by PBE. It focuses on changes about API usages and uses a few examples to infer such a synthesized program, in order to generate common patches and automatically apply them to other files.

Compared to existing PBE work, our TIDY approach indeed presents a PBE-based framework, without having to concretely synthesize a program. It aims to support smart transformations to maintain entity consistency for rich-formatted documents, which has not been well focused in the PBE research. As far as we know, although some existing work similarly aims to synthesize a transformer to maintain entity consistency [3,4,9] and handle entity relations [35–38]. They can hardly be applied to rich-formatted documents like PowerPoint, which usually includes few examples for program synthesis.

The closest work could be the one by Raza et al. [10], which proposed to synthesize a PBE-based program for handling structural transformations in rich-formatted documents with least general generalizations, and has implemented a plug-in module FlashFormat into PowerPoint. TIDY differs from this work. Unlike TIDY, which actively monitors users' operations at runtime and makes recommendations accordingly, FlashFormat calls for explicit invocation whenever users need assistance. This makes a fair comparison between TIDY and FlashFormat difficult, since it would rely highly on the provided examples for generating the two tools' internal artifacts (e.g., programs, models,

etc.) for follow-up recommendations, and the quality of these examples would be subject to how they are collected in different environments.

Blue-Pencil [39] is another piece of related work close to ours, which aims to identify qualified input–output examples through the user-editing history and make editing suggestions in practice. TIDY also differs from this work. First, TIDY and Blue-Pencil have different target applications. Although Blue-Pencil is proposed as a domain-agnostic approach, it emphasizes mostly on textual documents, such as C#, SQL, Markdown programs/documents, and spreadsheets. While our TIDY focuses on rich-formatted documents with explicit graphical interfaces, which significantly differ from textual documents. Second, after Blue-Pencil identifies suitable examples from multiple document versions, it would then invoke existing PBE engines to generate programs for later recommendation. In this sense, Blue-Pencil is more like an approach for example identification to be integrated with other PBE tools. Since we did not find available PBE engines for integrating with Blue-Pencil, a direct comparison between TIDY and Blue-Pencil for our targeted entity consistency tasks in rich-formatted documents can be infeasible. Third, Blue-Pencil makes efforts in proper example identification for later PBE by analyzing different versions of textual documents, for the reason that in textual documents, user intentions are typically vague and hidden. For rich-formatted documents (e.g., PowerPoint pages), they typically carry clear user intentions (although not explicitly presented), and thus TIDY makes attempts to dig them out by modeling common user intentions in its kernel goal library and conducting runtime goal matching. This approach can be more natural and suitable.

A typical feedback-driven process in the PBE field [40], usually refers to that synthesized PBE-based programs can be refined by feeding more additional inputs. TIDY somehow inherits a slightly different feedback-driven idea in its goal matching and selection, referring to TIDY's ability of gradually figuring out the exact user intention in analyzing more user operations. On one hand, TIDY may suggest multiple goals for matching when analyzing some user operations, and then when collecting more operations, TIDY would gradually generate more concrete goals for its next-operation recommendations. On the other hand, TIDY's goals for matching can be restricted back to one when its user clicks any of TIDY's recommended operations, indicating that the user has chosen to accept one specific goal. This makes TIDY's goal-matching continuously evolves according to its analyzed user operations and user's actions at runtime.

8. Conclusion

In this paper, we propose TIDY, a two-stage PBE-based framework, to assist automated entity transformations for their layout and style consistency in rich-formatted documents like PowerPoint, in a way adaptive to entity contexts and flexible with user selections. By examining entities' operation histories, it can automatically identify user intentions behind histories and make wise next-operation recommendations for users accordingly. Our experimental results show TIDY's effectiveness on both its stably promising hit rate (77.3% on average) and its marginal time overhead (7.2 ms on average).

There are still limitations in our work. For example, our goal library design has not covered all possible user intentions in practice and requires further extensions in future. TIDY's generalization for specific rich-formatted applications might need extra efforts. Our future work will focus on how to model more realistic user's intentions into TIDY's goal library, and apply it to more popular rich-formatted document applications.

CRediT authorship contribution statement

Shuguan Liu: Conceptualization, Methodology, Software, Writing - original draft, Investigation, Data curation. **Huiyan Wang:** Investigation, Validation, Writing - reviewing & editing. **Chang Xu:** Writing - reviewing & editing, Supervision, Project administration, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

We thank the editor and anonymous reviewers for their constructive suggestions. This work is supported by the National Natural Science Foundation of China (61932021, 61690204) and the Collaborative Innovation Center of Novel Software Technology and Industrialization.

References

[1] S. Gulwani, Programming by examples - and its applications in data wrangling, in: J. Esparza, O. Grumberg, S. Sickert (Eds.), Dependable Software Systems Engineering, in: NATO Science for Peace and Security Series - D: Information and Communication Security, vol. 45, IOS Press, 2016, pp. 137–158, http://dx.doi.org/10.3233/978-1-61499-627-9-137.

[2] S. Gulwani, O. Polozov, R. Singh, Program synthesis, Found. Trends Program. Lang. 4 (1–2) (2017) 1–119, http://dx.doi.org/10.1561/2500000010.

[3] V. Le, S. Gulwani, Flashextract: a framework for data extraction by examples, in: M.F.P. O'Boyle, K. Pingali (Eds.), ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '14, Edinburgh, United Kingdom - June 09 - 11, 2014, ACM, 2014, pp. 542–553, http://dx.doi.org/10.1145/2594291.2594333.

[4] S. Gulwani, Automating string processing in spreadsheets using input-output examples, in: T. Ball, M. Sagiv (Eds.), Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26-28, 2011, ACM, 2011, pp. 317–330, http://dx.doi.org/10.1145/1926385.1926423.

[5] R. Singh, S. Gulwani, Learning semantic string transformations from examples, Proc. VLDB Endow. 5 (8) (2012) 740–751, http://dx.doi.org/10.14778/2212351.2212356.

[6] I. Drosos, T. Barik, P.J. Guo, R. DeLine, S. Gulwani, Wrex: A unified programming-by-example interaction for synthesizing readable code for data scientists, in: R. Bernhaupt, F.F. Mueller, D. Verweij, J. Andres, J. McGrenere, A. Cockburn, I. Avellino, A. Goguey, P. Bjøn, S. Zhao, B.P. Samson, R. Kocielnik (Eds.), CHI '20: CHI Conference on Human Factors in Computing Systems, Honolulu, HI, USA, April 25-30, 2020, ACM, 2020, pp. 1–12, http://dx.doi.org/10.1145/3313831.3376442.

[7] R. Singh, S. Gulwani, Predicting a correct program in programming by example, in: D. Kroening, C.S. Pasareanu (Eds.), Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I, in: Lecture Notes in Computer Science, vol. 9206, Springer, 2015, pp. 398–414, http://dx.doi.org/10.1007/978-3-319-21690-4_23.

[8] R. Singh, S.G. 2016., Transforming spreadsheet data types using examples, ACM SIGPLAN Not. 51 (1) (2016) 343–356.

[9] N. Yaghmazadeh, C. Klinger, I. Dillig, S. Chaudhuri, Synthesizing transformations on hierarchically structured data, in: C. Krintz, E. Berger (Eds.), Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2016, Santa Barbara, CA, USA, June 13-17, 2016, ACM, 2016, pp. 508–521, http://dx.doi.org/10.1145/2908080.2908088.

[10] M. Raza, S. Gulwani, N. Milic-Frayling, Programming by example using least general generalizations, in: C.E. Brodley, P. Stone (Eds.), Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, QuÉBec City, QuÉBec, Canada, AAAI Press, 2014, pp. 283–290.

[11] R. Bhatia, Matrix Analysis, Springer, Berlin, 1996.

[12] T.Y. Tso, S. Tweedie, Planned extensions to the linux ext2/ext3 filesystem, 2002, pp. 235–243.

[13] M. Ceci, D. Malerba, Hierarchical classification of HTML documents with webclassii, 2003, pp. 57–72.

[14] PPT Object-model, https://docs.microsoft.com/en-us/office/vba/api/overview/powerpoint/object-model.

[15] D. Gossot, Y. Miaux, A. Guermazi, M. Celerier, J. Frija, The hook-wire technique for localization of pulmonary nodules during thoracoscopic resection, Chest 105 (5) (1994) 1467–1469.

[16] MS Office Forum, https://www.msofficeforums.com/.

[17] S. Srivastava, S. Gulwani, J.S. Foster, From program verification to program synthesis, in: M.V. Hermenegildo, J. Palsberg (Eds.), Proceedings of the 37th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2010, Madrid, Spain, January 17-23, 2010, ACM, 2010, pp. 313–326, http://dx.doi.org/10.1145/1706299.1706337.

[18] S. Itzhaky, S. Gulwani, N. Immerman, M. Sagiv, A simple inductive synthesis methodology and its applications, in: W.R. Cook, S. Clarke, M.C. Rinard (Eds.), Proceedings of the 25th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2010, October 17-21, 2010, Reno/Tahoe, Nevada, USA, ACM, 2010, pp. 36–46, http://dx.doi.org/10.1145/1869459.1869463.

[19] S. Gulwani, S. Jha, A. Tiwari, R. Venkatesan, Synthesis of loop-free programs, in: M.W. Hall, D.A. Padua (Eds.), Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2011, San Jose, CA, USA, June 4-8, 2011, ACM, 2011, pp. 62–73, http://dx.doi.org/10.1145/1993498.1993506.

[20] R.E. Brooks, "Watch what i do: Programming by demonstration, " edited by allen cypher (book review), Int. J. Man-Mach. Stud. 39 (6) (1993) 1051–1057.

[21] H. Lieberman (Ed.), Your Wish is My Command, in: The Morgan Kaufmann Series in Interactive Technologies, Morgan Kaufmann / Elsevier, 2001, http://dx.doi.org/10.1016/b978-1-55860-688-3.x5000-3.

[22] Y. Shen, K. Chakrabarti, S. Chaudhuri, B. Ding, L. Novik, Discovering queries based on example tuples, in: C.E. Dyreson, F. Li, M.T. Özsu (Eds.), International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014, ACM, 2014, pp. 493–504, http://dx.doi.org/10.1145/2588555.2593664.

[23] S. Cheema, S. Buchanan, S. Gulwani, J.J.L. Jr., A practical framework for constructing structured drawings, in: T. Kuflik, O. Stock, J.Y. Chai, A. Krüger (Eds.), 19th International Conference on Intelligent User Interfaces, IUI 2014, Haifa, Israel, February 24-27, 2014, ACM, 2014, pp. 311–316, http://dx.doi.org/10.1145/2557500.2557522.

[24] N. Meng, M. Kim, K.S. McKinley, LASE: locating and applying systematic edits by learning from examples, in: D. Notkin, B.H.C. Cheng, K. Pohl (Eds.), 35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013, IEEE Computer Society, 2013, pp. 502–511, http://dx.doi.org/10.1109/ICSE.2013.6606596.

[25] J. Jacobellis, N. Meng, M. Kim, LASE: an example-based program transformation tool for locating and applying systematic edits, in: D. Notkin, B.H.C. Cheng, K. Pohl (Eds.), 35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013, IEEE Computer Society, 2013, pp. 1319–1322, http://dx.doi.org/10.1109/ICSE.2013.6606707.

[26] A. Leung, J. Sarracino, S. Lerner, Interactive parser synthesis by example, in: D. Grove, S. Blackburn (Eds.), Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation, Portland, OR, USA, June 15-17, 2015, ACM, 2015, pp. 565–574, http://dx.doi.org/10.1145/2737924.2738002.

[27] S. Gulwani, P. Jain, Programming by examples: PL meets ML, in: B.E. Chang (Ed.), Programming Languages and Systems - 15th Asian Symposium, APLAS 2017, Suzhou, China, November 27-29, 2017, Proceedings, in: Lecture Notes in Computer Science, vol. 10695, Springer, 2017, pp. 3–20, http://dx.doi.org/10.1007/978-3-319-71237-6_1.

[28] D.W. Barowy, S. Gulwani, T. Hart, B.G. Zorn, Flashrelate: extracting relational data from semi-structured spreadsheets using examples, in: D. Grove, S. Blackburn (Eds.), Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation, Portland, OR, USA, June 15-17, 2015, ACM, 2015, pp. 218–228, http://dx.doi.org/10.1145/2737924.2737952.

[29] S. Gulwani, W.R. Harris, R. Singh, Spreadsheet data manipulation using examples, Commun. ACM 55 (8) (2012) 97–105, http://dx.doi.org/10.1145/2240236.2240260.

[30] S. Gulwani, M. Marron, Nlyze: interactive programming by natural language for spreadsheet data analysis and manipulation, in: C.E. Dyreson, F. Li, M.T. Özsu (Eds.), International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014, ACM, 2014, pp. 803–814, http://dx.doi.org/10.1145/2588555.2612177.

[31] R.R. de Sousa, G. Soares, L. D'Antoni, O. Polozov, S. Gulwani, R. Gheyi, R. Suzuki, B. Hartmann, Learning syntactic program transformations from examples, 2016, CoRR abs/1608.09000. arXiv:1608.09000.

[32] R. Rolim, G. Soares, L. D'Antoni, O. Polozov, S. Gulwani, R. Gheyi, R. Suzuki, B. Hartmann, Learning syntactic program transformations from examples, in: S. Uchitel, A. Orso, M.P. Robillard (Eds.), Proceedings of the 39th International Conference on Software Engineering, ICSE 2017, Buenos Aires, Argentina, May 20-28, 2017, IEEE / ACM, 2017, pp. 404–415, http://dx.doi.org/10.1109/ICSE.2017.44.

[33] J. Andersen, J.L. Lawall, Generic patch inference, in: 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE 2008), 15-19 September 2008, L'Aquila, Italy, IEEE Computer Society, 2008, pp. 337–346, http://dx.doi.org/10.1109/ASE.2008.44.

[34] J. Andersen, J.L. Lawall, Generic patch inference, Autom. Softw. Eng. 17 (2) (2010) 119–148, http://dx.doi.org/10.1007/s10515-010-0062-z.

[35] J. Wu, Y. Jiang, C. Xu, S. Cheung, X. Ma, J. Lu, Synthesizing relation-aware entity transformation by examples, in: M. Chaudron, I. Crnkovic, M. Chechik, M. Harman (Eds.), Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018, ACM, 2018, pp. 266–267, http://dx.doi.org/10.1145/3183440.3194963.

[36] S. Zhang, Y. Sun, Automatically synthesizing SQL queries from input-output examples, in: E. Denney, T. Bultan, A. Zeller (Eds.), 2013 28th IEEE/ACM International Conference on Automated Software Engineering, ASE 2013, Silicon Valley, CA, USA, November 11-15, 2013, IEEE, 2013, pp. 224–234, http://dx.doi.org/10.1109/ASE.2013.6693082.

[37] C. Wang, A. Cheung, R. Bodík, Synthesizing highly expressive SQL queries from input-output examples, in: A. Cohen, M.T. Vechev (Eds.), Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2017, Barcelona, Spain, June 18-23, 2017, ACM, 2017, pp. 452–466, http://dx.doi.org/10.1145/3062341.3062365.

[38] A. Abouzied, D. Angluin, C.H. Papadimitriou, J.M. Hellerstein, A. Silberschatz, Learning and verifying quantified boolean queries by example, in: R. Hull, W. Fan (Eds.), Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2013, New York, NY, USA - June 22 - 27, 2013, ACM, 2013, pp. 49–60, http://dx.doi.org/10.1145/2463664.2465220.

[39] A. Miltner, S. Gulwani, V. Le, A. Leung, A. Radhakrishna, G. Soares, A. Tiwari, A. Udupa, On the fly synthesis of edit suggestions, Proc. ACM Program. Lang. 3 (OOPSLA) (2019) 143:1–143:29, http://dx.doi.org/10.1145/3360569.

[40] X. Gao, S. Barke, A. Radhakrishna, G. Soares, S. Gulwani, A. Leung, N. Nagappan, A. Tiwari, Feedback-driven semi-supervised synthesis of program transformations, Proc. ACM Program. Lang. 4 (OOPSLA) (2020) 219:1–219:30, http://dx.doi.org/10.1145/3428287.