Front. Comput. Sci., 2021, 15(5): 155209
https://doi.org/10.1007/s11704-020-9518-x

RESEARCH ARTICLE

A semi-transparent selective undo algorithm for multi-user collaborative editors

Weiwei CAI[1], Fazhi HE (✉)[1], Xiao LV[2], Yuan CHENG[3]

1 School of Computer Science, Wuhan University, Wuhan 430072, China
2 Department of Computer Engineering, Naval University of Engineering, Wuhan 430072, China
3 School of Information Management, Wuhan University, Wuhan 430072, China

Abstract Multi-user collaborative editors are useful computer-aided tools to support human-to-human collaboration. For multi-user collaborative editors, selective undo is an essential utility enabling users to undo any editing operations at any time. Collaborative editors usually adopt operational transformation (OT) to address concurrency and consistency issues. However, it is still a great challenge to design an efficient and correct OT algorithm capable of handling both normal do operations and user-initiated undo operations because these two kinds of operations can interfere with each other in various forms. In this paper, we propose a semi-transparent selective undo algorithm that handles both do and undo in a unified framework, which separates the processing part of do operations from the processing part of undo operations. Formal proofs are provided to prove the proposed algorithm under the well-established criteria. Theoretical analysis and experimental evaluation are conducted to show that the proposed algorithm outperforms the prior OT-based selective undo algorithms.

Keywords human-centric collaboration, collaborative editing systems, selective undo, concurrency control, replication consistency

1 Introduction

Human-to-human collaboration is becoming more and more critical in modern highly dispersed organizations and teams. Computer-supported collaborative systems enable geographically separated people working together to achieve a common goal through computer networks [1–11]. These systems provide strong technical supports for human-to-human collaboration [12–15].

Multi-user collaborative editors (co-editors) are a typical kind of collaborative systems. Co-editors allow users to freely edit shared and replicated documents in a non-blocking way at the same time [2, 6, 8, 16–20]. A general consistency requirement of co-editors is the operation effect preservation and convergence preservation [2, 8, 16–18]. Operational transformation (OT) can well satisfy the consistency requirements of co-editors in unconstrained collaboration environments [2, 8, 16–18]. As an optimistic concurrency control approach, OT coordinates concurrent editing operations by transforming and executing the operations in the transformed forms rather than the original forms. Transformation of operations is the key for OT technique to support a wide range of collaborative applications [2, 8, 9, 16–18].

Undo is a practical and powerful utility in interactive applications [21–30]. Undo utilities can help users revert unwanted actions and recover from errors. Most single-user editors provide a linear undo utility, which allows users to undo the last operation in chronological order. However, in unconstrained collaboration environments, the operations may be concurrently generated by multiple users and executed in arbitrary orders at different cooperating sites. The globally last operation may be generated by other users, and the locally last operation may not be the globally last operation. A suitable undo model for such complicated interactions is non-linear undo, called *selective undo* or *any undo* [22, 23, 25–28], which allows operations to be undone in any orders.

Several research work extended the OT-based consistency maintenance algorithms to support both do and undo operations in co-editors [23, 25–28]. Most solutions treat an undo command as an inverse operation and transform the inverse operation like a normal editing operation. However, normal editing operations and inverse operations in essence have different semantics and could interfere with each other, which complicates the integrated OT solutions capable of supporting both do and undo in correctness and efficiency.

In our view, the most critical reason for the complication is that the existing solutions tightly couple undo-related problems with do-related problems. We propose a semi-transparent selective undo algorithm to reduce the complication by decoupling undo-related problems from do-related problems. The problem separation needs to maintain a kind of metadata of shared documents on which the execution effect of undo operations has no influence on the execution effect of do operations. Our solution is a semi-transparent approach in the sense that the solution can transparently reuse the existing do algorithms to ensure the consistency of the metadata and introduce new components to achieve the consistency of the shared documents by the consis-

Received December 24, 2019; accepted June 9, 2020

E-mail: fzhe@whu.edu.cn

tent metadata. The main contributions are as follows:

1) We present an efficient and correct undo algorithm that supports both do and undo operations in co-editors,
2) We prove the proposed algorithm with formal proofs under the well-established correctness criteria [23, 25–28],
3) We conduct a comprehensive comparison of the proposed algorithm and the prior algorithms with both theoretical analysis and experimental evaluation.

The rest of this paper is structured as follows. Section 3 introduces the basic concepts of OT technique and reviews the prior work. Section 4 illustrates the proposed algorithm ST-Undo. Section 5 discusses the operation migration scheme. Section 6 gives a working example of ST-Undo. Section 7 compares ST-Undo with the prior approaches. The last section summaries contributions and future directions.

2 Background and related work

2.1 OT supported co-editors

Co-editors generally adopt replicated storage, i.e., each cooperating site maintains a copy of shared documents [16–20]. Users can freely edit any part of local replicas. Updates (or operations) on the replicas are exchanged over networks among cooperating sites. As the operations are allowed to be executed in different orders at different replicas, distributed replicas may be inconsistent.

To address the consistency issues in such unconstrained collaboration environments, the operational transformation (OT) approach allows *local operations* (the operations generated at local sites) to be executed immediately but requires *remote operations* (the operations generated at remote sites) to be transformed before their execution [16,27,31–33]. An OT algorithm generally consists of two components: transformation control algorithms and transformation functions.

Transformation control algorithms maintain a data structure called *history buffer*, which records a set of executed operations at each cooperating site [16, 27, 31–33]. Given a remote operation, according to *causal/concurrent relations* and *context relations* [16, 27, 34], the transformation control algorithms transform the remote operation against its concurrent operations in the history buffer in a certain order.

Transformation functions are responsible for doing the operation transformation that may modify the parameters of an operation and change its execution forms [35–39]. They are dependent on the semantic of operations. Function $T(O_a, O_b)$ transforms O_a against O_b to get a transformed version of O_a, denoted as O'_a. For convenience, $LT(O, L)$ represents the transformation of O with a sequence of operations L one by one. Transformation functions need to cooperate with transformation control algorithms and satisfy some properties, e.g., TP1 and TP2 [31].

Definition 1 (Transformation Property, TP1). *Given two operations O_a and O_b, generated from a state S, a transformation function T satisfies TP1 if and only if the effect of executing O_a before O'_b is the same as the effect of executing O_b before O'_a, i.e., $S \cdot O_a \cdot O'_b = S \cdot O_b \cdot O'_a$, where $O'_a = T(O_a, O_b)$ and $O'_b = T(O_b, O_a)$.*

Definition 2 (Transformation Property, TP2). *Given three operations O_a, O_b, and O_c, generated from the same state, a transformation function T satisfies TP2 if and only if the transformation of O_c against O_a before O'_b gets the same result as the transformation of O_c against O_b before O'_a, i.e., $LT(O_c, [O_a, O'_b]) = LT(O_c, [O_b, O'_a])$, where $O'_a = T(O_a, O_b)$ and $O'_b = T(O_b, O_a)$.*

Following the established conventions [16, 35–41], a shared text document is abstracted as a sequence of characters (or objects). Every character has a positional index. Updates on the documents can be represented by two primitive operations: $I(p, c)$ inserts character c at position p and $D(p, c)$ deletes character c at position p.

The scenario depicted in Fig. 1 illustrates the basic idea of OT. Initially, the replicated documents at site 1 and 2 have the same state, $S_{10} = S_{20}$ ="a". Then User A and User B concurrently generate two operations $O_1 = I(1, y)$ and $O_2 = I(0, x)$, respectively. The transformation and execution of O_1 and O_2 at the cooperating sites are described as follows:

- at site 1, when O_2 is received, the document state is S_{11} ="ay" and the history buffer is $H_{11} = [O_1]$; O_2 is transformed by $T(O_2, O_1)$; as O_1 has no impact on the effect position of O_2, the transformation result is $O'_2 = I(0, x)$; the document state becomes S_{12} ="xay";
- at site 2, when O_1 is received, the document state is S_{21} ="xa" and the history buffer is $H_{21} = [O_2]$; O_1 is transformed by $T(O_1, O_2)$; as O_2 shifts the effect position of O_1 to the right by one position, the transformation result is $O'_1 = I(2, y)$; the document state becomes S_{22} ="xay".

2.2 Correctness criteria of selective undo

Users of interactive applications are familiar with the undo functionality: undoing an operation is to eliminate the effect of this operation and restore the document to a previous state at the time that the last operation was not executed before. Multi-user collaborative editors should also provide a similar undo effect that is easy-to-understand and reasonable to collaborative users. The undo effect in Definition 3 is a general specification of what undo effect should be achieved by undo algorithms in collaboration environments [23]. This specification has guided the design of undo algorithms for co-editors in the literature [23, 25–28, 30]. Our undo algorithm also aims to

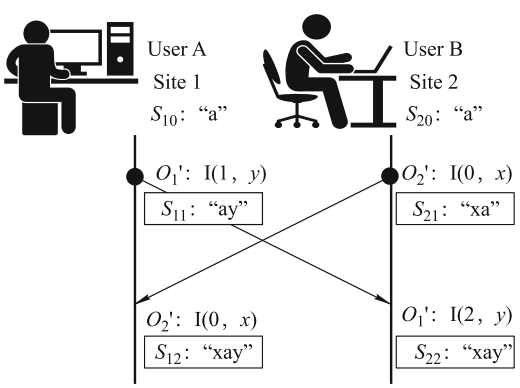


Fig. 1 Concurrent operations are commutative by transforming operations

achieve such undo effect.

Definition 3 (Undo Effect). *Undoing an operation would remove the effect of its own but retain the effects of the other operations.*

We introduce an example to illustrate the undo effect. The user at site 1 generates an operation $O_1 = D(1, c)$ on S_{10} ="ac" to get S_{11}="a". Concurrently, the user at site 2 generates an operation $O_2 = I(1, b)$ on S_{20} ="ac" to get S_{21}="abc". When O_2 is received at site 1, O_2 is transformed and executed as $O'_2 = I(1, b)$, and the document becomes S_{12}="ab". When O_1 is received at site 2, O_1 is transformed and executed as $O'_1 = D(2, c)$, and the document becomes S_{22}="ab". According to Definition 3, the undo effect of O_1 should restore the document to "abc", which removes the effect of O_1 but retains the effect of O_2. At site 2, the desired effect of undoing O_1 can be achieved by applying $I(2, c)$ (the inverse operation of O'_1) on S_{22}. However, at site 1, applying $I(1, c)$ (the inverse operation of O'_1) on S_{12} fails to recover the document state "abc". As O_1 is not the last operation and O'_2 has been executed after O_1, the inverse operation cannot remove the effect of O_1.

2.3 Interference between do and undo

In the OT-based systems that transform both do and undo operations, the same and different types of operations interfere with each other in various forms [23]. One of the most thorny problems in handling both do and undo is that the ordering relations between concurrently inserted objects may abnormally be reversed in co-editing scenarios [35–40].

Figure 2 illustrates the abnormal ordering problem, which is developed from the scenario depicted in Fig. 1. The difference is that three users collaboratively edit a shared document and User C at site 3 generates an operation O_3 from the initial document state. The transformation and execution of these operations are illustrated as follows:

- at site 1, when O_3 is received, the history buffer is $[O_1, O'_2]$; O_3 is transformed by $LT(O_3, [O_1, O'_2])$; the transformation result is $O'_3 = D(1, a)$, the document state becomes S_{13} ="xy";
- at site 2, when O_3 is received, the history buffer is $[O_2, O'_1]$; O_3 is transformed by $LT(O_3, [O_2, O'_1])$; the transformation result is $O'_3 = D(1, a)$, the document state becomes S_{23} ="xy";
- at site 3, when O_1 is received, the document state is S_{31}=" " and the history buffer is $H_{31} = [O_3]$; O_1 is transformed by $T(O_1, O_3)$; as O_3 has shifted the effect position of O_1 to the left by one position, the transformation result is $O'_1 = I(0, y)$; when O_2 is received, the document state is S_{32} ="y" and the history buffer is $H_{32} = [O_3, O'_1]$; O_2 is transformed by $LT(O_2, [O_3, O'_1])$; as O_3 has no impact on the effect position of O_2, the transformation result of $T(O_2, O_3)$ is $O'_2 = I(0, x)$, which has a position tie with O'_1; according to the tie-breaking policy[1)], the transformation result of $T(O'_2, O'_1)$ may be $O'_2 = I(1, x)$; the document state becomes S_{33} ="yx".

[1)] The position tie is usually broken by site identifiers.

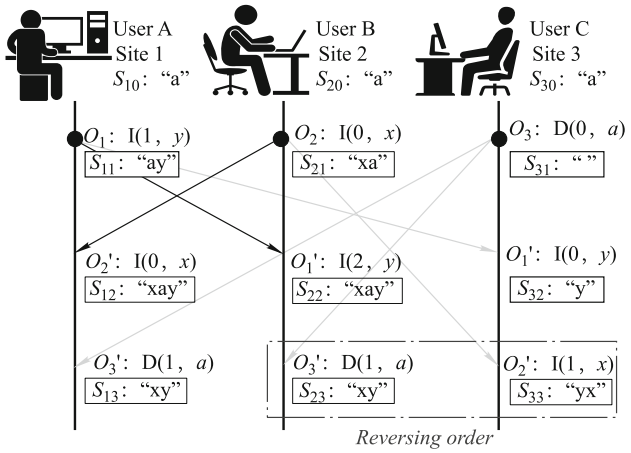


Fig. 2 Reversing the preceding order of objects

In the above scenario, the cooperating sites have different document states after executing the same set of operations in different orders. The inconsistency is obviously not acceptable. To address the inconsistency issue, most OT algorithms impose a transformation order that is consistent with total order relations among operations [33]. Even though convergent results can be achieved by constraining transformation orders, the results may not preserve the preceding orders of concurrently inserted objects. As shown in Fig. 2, the final results may converge to either "xy" or "yx", due to the uncertainty of total ordering relations among operations. It is clear that the state "yx" is strange and unreasonable to collaboration users, because "y" would be definitely placed before "x" if O_3 has not been generated by User C at site 3.

The abnormal ordering issue causes problems in handling undo operations. Still consider the above example. If users want to undo O_3 at a cooperating site after the three operations have been applied on all the replicas, the correct undo effect should restore the document state "xay" for removing the effect of O_3 but retaining the effects of O_1 and O_2. However, if the replicas converge to "yx", it is almost impossible to get the desired document state "xay" after O_3 is undone.

2.4 Overview of related work

In the literature, there are two highly researched concurrency control approaches for co-editing systems: operational transformation (OT) [42–44] and commutative replicated data type (CRDT) [45–48].

2.4.1 Operational transformation

Early OT-based undo solutions transform inverse operations in the same way as do operations according to causal/concurrent/context relations [23–28]. These solutions have high computation cost in avoiding the necessary inverse properties on the transformation functions. AnyUndo [23] enforces a do-undo pair to behave like an identity operation by coupling and decoupling pairs of do and undo operations. The time complexity for undo in AnyUndo is exponential in the number of operations in the history buffer. COT [27] explicitly represents causal/concurrent/context relations between a variety of operations by context vector timestamps. The time com-

plexity of COT in handling undo operations is exponential in the number of operations in the history buffer. By storing N versions of a normal do operation, where N is the number of cooperating sites, COT can achieve a linear time complexity in handling undo operations. SOCT-Undo [24] works under the assumption that there is a transformation function capable of satisfying the inverse properties. The time complexity of SOCT-Undo is quadratic in the number of operations in the history buffer.

The follow-up OT-based undo solutions invent functional components to address the abnormal ordering problem. TTF [36] introduces an object sequence to keep deleted objects. ABT [39] introduces a special history buffer in which insert operations are placed before delete operations. UNO [25,26] is a selective undo algorithm built on TTF. As deleted objects are never lost, UNO can preserve the ordering relations among objects. Except for the object sequence, UNO stores both do and undo operation in the history buffer. The time and space complexity of UNO is linear in the size of the object sequence plus the number of operations in the history buffer. ABTU [28] is developed from ABT [39]. In ABTU, undo operations are stored in the form of inverse operations of the corresponding do operations in the history buffer. As an operation may be transformed with both do and undo operations, ABTU arranges the operations in the history buffer according to their effect positions. ABTU has a linear time and space complexity in the size of history buffer.

2.4.2 Commutative replicated data type

CRDT addresses the abnormal ordering issue without requiring transformation functions [45–48]. CRDT designs a replicated data type on which the operations having concurrent relations must be commutative. In other words, the concurrent operations can lead to the same abstract state of the data type in any execution orders. Specific to collaborative text editing, a CRDT solution is a sequence data type capable of supporting the addition and removal of elements.

In the past decade, several CRDT-based undo solutions were invented [46–48]. Those algorithms have common features: recording operation effects in an object sequence, associating objects with unique identifiers, searching objects by identifiers, counting visible or invisible objects, and manipulating the object sequence by identifiers-based operations. To guarantee the consistency of object sequences, the CRDT-based solutions require the identifier-based operations with concurrent relations to be commutative. However, achieving the consistency of the object sequences is not the final target. The CRDT-based solutions must convert the identifier-based operations to/from the position-based operations to finally achieve the consistency of shared and replicated documents. The time and space overhead of the CRDT-based solutions are dependent on the size of the object sequences because the conversion must update and search the object sequences [43].

2.4.3 Comparison of OT and CRDT

According to the recent research studies [42–44], OT and CRDT both adopt the strategy of transforming operations to solve the consistency problems in collaborative applications. They are both a kind of transformation-based concurrency control approaches but different in the way of transforming operations.

UNO [25,26] is an OT-based solution, but it has some typical features of the CRDT-based solutions [45–48]. UNO requires an object sequence to keep tombstones and position-based operations to manipulate the object sequence. UNO achieves the consistency of the object sequence by the combination of transformation control algorithms and TTF transformation function. However, to ensure the consistency of shared documents, UNO must do the conversion between the operations manipulating the object sequence and the operations manipulating the shared documents. Thus, UNO has the same time and space complexity as the CRDT-based solutions. They both are dependent on the size of the object sequences. As UNO does not need to associate the complicated identifiers with objects, UNO could be more efficient than the CRDT-based solutions in space overhead.

3 The proposed algorithm

Keeping deleted characters can avoid the abnormal ordering issue. ST-Undo stores an intermediate-layer document as the metadata of shared documents. The metadata consists of visible or invisible objects, corresponding to the characters ever appearing in shared documents. The undo effect of an operation on intermediate-layer documents only changes the visible attribute of an object, without impacting its relative position in the intermediate-layer documents. Thus, undoing operations do not influence the transformation of normal do operations, ST-Undo can individually address do-related problems and undo-related problems at the intermediate-layer data model. This section will illustrate the high-level workflow and low-level functional components and how they interact with the metadata to achieve the desired consistency and undo effect.

3.1 Two-layer operation and data model

Intermediate-layer documents are introduced as the metadata of shared documents. From the perspective of applications, the characters in a shared document are ordered and indexed in a linear addressing space. The corresponding intermediate-layer document records all the characters that ever have appeared in the shared document, including the deleted characters. We refer the application-layer documents to the shared documents and refer the application-layer operations to the primitive character-wise insert and delete.

An intermediate-layer document can be considered to be a sequence of visible and invisible objects. Each object C has two attributes: $Num(C)$ represents its visible attribute, and $Char(C)$ records a normal character c. $Num(C)$ records how many times an object C is deleted by operations, which could be normal do operations or undo operations. Given an object C, before any operation targets it, we have $Num(C) = 0$. Every time C is deleted, its visible attribute value is correspondingly updated, $Num(C) = Num(C) + 1$. If $Num(C) > 0$, C is an invisible object; otherwise C is a visible object. Two operations update an intermediate-layer document: $I(p, c)$ inserts a visible object C at position p and $M(p, c, v)$ marks the object at position p visible if $v = -1$ or invisible if $v = +1$.

Definition 4 (Effect of Intermediate-layer Insert). *Given an*

operation $IO = I(p, c)$ *defined on a state* IS*, the effect of applying* IO *on* IS *is expressed as* $IS \cdot I(p, c) = IS[0, p-1] + C + IS[p, \ldots]$*, where* $Num(C) = 0$ *and* $Char(C) = c$.

Definition 5 (Effect of Intermediate-layer Mark). *Given an operation* $IO = M(p, c, v)$ *defined on a state* IS*, the effect of applying* IO *on* IS *is expressed as* $IS \cdot M(p, c, v) = IS[0, p-1] + C' + IS[p+1, \ldots]$*, where* $Num(C') = Num(IS[p]) + v$.

Definitions 4 and 5 give formal specifications of the effects of intermediate-layer operations. It can be observed that an intermediate-layer insert has a similar shifting effect as an application-layer insert but an intermediate-layer mark operation has no shifting effect and only updates the visible attribute of an object without moving objects.

The mapping relationship between an application-layer document S and an intermediate-layer document IS can be described as follows:

- $S[p] \mapsto IS[p+m]$, where $IS[p+m]$ is the pth visible object and m is the number of invisible objects before $IS[p+m]$;
- $IS[p] \mapsto S[p-m]$, where m is the number of invisible objects before the visible object $IS[p]$.

3.2 Functional components and workflow

ST-Undo consists of four major functional components: a bi-directional operation migration scheme, a transformation function for intermediate-layer operations, a transformation control algorithm, and an undo algorithm. They work together to handle do and undo operations.

Users at cooperating sites can concurrently edit the replicas of shared documents. The changes made on the local replicas are detected and delivered as application-layer operations to the underlying ST-Undo algorithm. At the same time, ST-Undo algorithm receives update information from other sites. It delivers transformed application-layer operations to co-editors for reflecting other users' actions. Figure 3 illustrates the workflow of an application-layer operation at two cooperative sites,

- at site 1, the operation migration scheme first migrates O to an intermediate-layer operation IO, and applies IO on the intermediate-layer document IS_1; finally IO is sent to site 2 over networks;
- at site 2, the transformation control algorithm first transforms the received operation IO against its concurrent operations that have been executed at site 2, and the operation migration scheme migrates the transformed operation IO' back to an application-layer operation O' and applies IO' on IS_2; finally the user at site 2 is noticed of O after O' is applied on S_2.

Users at cooperating sites can undo any operation. As every application-layer operation corresponds to one intermediate-layer operation, undoing an application-layer operation is equivalent to undoing an intermediate-layer operation. ST-Undo algorithm accepts an undo command $Undo(IO)$ and delivers an application-layer operation O_x to co-editor for removing the effect of O. Figure 4 illustrates the workflow of an undo command at two cooperative sites,

- at site 1, the undo algorithm computes an operation IO_x that can undo the effect of IO on IS_1, then the operation migration scheme migrates IO_x to an application-layer operation O_x and applies IO_x on IS_1; finally the user is noticed of the undo effect of O after O_x is applied on S_1; the undo command $Undo(IO)$ is sent to site 2;
- at site 2, the undo command is received and ready for execution, the undo algorithm computes an operation IO_x that can undo the effect of IO on IS_2, then the operation migration scheme migrates IO_x to an application-layer operation O_x and applies IO_x on IS_2; finally the user is noticed of the undo effect of O after O_x is applied on S_2.

In an ST-Undo system, there are at least two concurrent and exclusive threads. One thread handles local do and undo operations, and the other thread handles remote do and undo operations. At a time, only one thread that performs the operation migration procedure is running in the system.

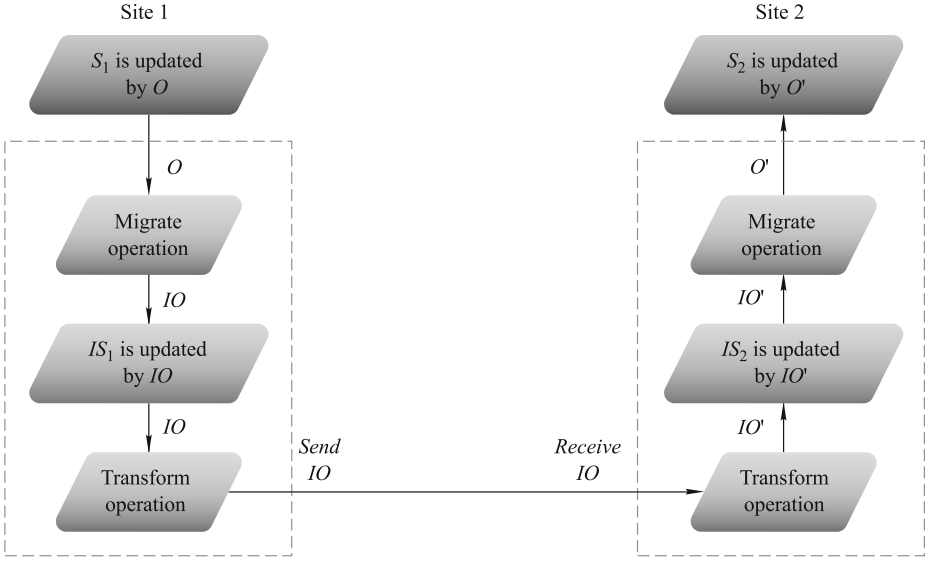


Fig. 3 Workflow of do operations

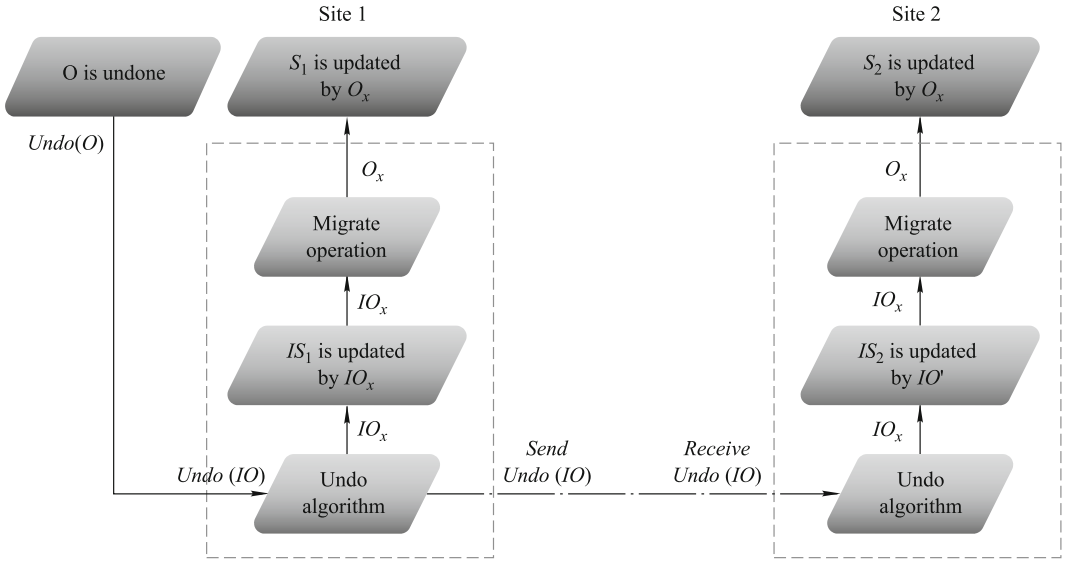


Fig. 4 Workflow of undo operations

3.3 Supporting normal do operations

To be capable of handling do operations, ST-Undo chooses the combination of an existing transformation control algorithm and a transformation function specific to intermediate-layer operations. There are several transformation control algorithms, such as TIBOT [32], COT [27], POT [33], which are independent of the specific operation and data models and only require the transformation functions capable of preserving TP1. Thus, ST-Undo can guarantee the consistency of intermediate-layer documents by TP1-preserving transformation functions and TP2-avoiding transformation control algorithms. We introduce a transformation function for intermediate-layer operations capable of satisfying TP1.

Some notations are introduced for the description of the transformation function. Given an intermediate-layer operation IO, $Type(IO)$ denotes the operation type: I for insert operations and M for mark operations, $Pos(IO)$ denotes the positional index, $Sid(IO)$ denotes the unique identifier of the cooperating site at which IO is generated.

Algorithm 1 describes a transformation function that transforms a pair of intermediate-layer operations. If IO_2 is a mark operation, the resulting operation $IO_1' = T(IO_1, IO_2)$ is the same as the original operation, as IO_2 has no impact on IO_1. However, if IO_2 is an insert operation, the resulting operation $IO_1' = T(IO_1, IO_2)$ may change the position attribute to consider the shifting effect of IO_2 on IO_1.

Algorithm 1 T(IO_1,IO_2)

Input: IO_1 and IO_2 are intermediate-layer ops
Output: IO_1' is a transformed version of IO_1

```
1:  IO1' ← IO1;
2:  if Type(IO1) = I ∧ Type(IO2) = I then
3:    if Pos(IO1) > Pos(IO2) then
4:      Pos(IO1') ← Pos(IO1') + 1;
5:    els if Pos(IO1) = Pos(IO2) ∧ Sid(IO1) > Sid(IO2) then
6:      Pos(IO1') ← Pos(IO1') + 1;
7:    end if
8:  els if Type(IO1) = M ∧ Type(IO2) = I then
9:    if Pos(IO1) ⩾ Pos(IO2) then
10:     Pos(IO1') ← Pos(IO1') + 1;
11:   end if
12: end if
13: Return IO1';
```

3.4 Supporting selective undo

The undo algorithm shares the history buffer with the do algorithm for the operation selection and undo. However, the undo algorithm does not add new operations, remove and transform the existing operations in the history buffer. Do and undo operations do not interfere with each other in transformation, thus the undo algorithm can transparently work together with the do algorithm.

An undo command $Undo(IO)$ indicates the selected operation to be undone. As every operation can be uniquely identified by a site identifier and a sequence number, it is trivial to determine which operation in the history buffer should be undone. ST-Undo handles an undo command in a unified way, no matter whether it is issued at local or remote cooperating sites. Given an undo command $Undo(IO)$, ST-Undo would perform the following steps:

1) determine whether IO should be undone or redone,
2) transform a mark operation IO_x against the operations executed or generated after IO to get IO_x', where IO_x' is able to cancel the effect of IO,
3) update the intermediate-layer document by the transformed operation IO_x' and migrate IO_x' to an application-layer operation O_x.

At the first step, ST-Undo first checks the number of undo commands performed on the operation to address multi-undo cases. In collaborative environments, an operation may be undone several times by sequential or concurrent operations. In these cases, ST-Undo handles an undo command in the odd-even-switch way [27]: if an operation is undone for odd times, the operation is undone; if an operation is undone for even

times, the operation is redone.

At step 2, ST-Undo computes an operation IO'_x whose execution effect can cancel the effect of IO on the intermediate-layer document. First, an operation IO_x is generated to remove the effect of IO on the document state at the time of executing IO. Between the execution of IO and the undo of IO, a sequence of operations L may have been applied on the document. To consider the effects of those operations, ST-Undo transforms IO_x against L to get an operation $IO'_x = LT(IO_x, L)$, which can remove the effect of IO on the document state at the time of undoing IO.

At step 3, ST-Undo removes the effect of IO by applying IO'_x on the intermediate-layer document. As the state changes of the intermediate-layer document need to be synchronized with the application-layer document, IO'_x would be migrated to an application-layer operation O_x, which can remove the effect of O on the current application-layer document.

3.5 Correctness

We formally prove that ST-Undo can maintain the consistency of replicated documents and acquire the correct undo effect. ST-Undo can guarantee the consistency by the combination of a TP1-preserving transformation function and an existing TP2-avoiding transformation control algorithm.

Lemma 1 The transformation function T given in Algorithm 1 satisfies TP1.

Proof Given a pair of intermediate-layer operations IO_1 and IO_2 generated on a state IS, we prove that $IS \cdot IO_1 \cdot IO'_2 = IS \cdot IO_2 \cdot IO'_1$. Consider the three combinations of different operation types,

1) $IO_1 = I(p_1, c_1)$ and $IO_2 = I(p_2, c_2)$, the position parameters have four relations:
   (a) $p_1 > p_2$, the left side is $IS[0, p_2 - 1] + C_2 + IS[p_2, p_1 - 1] + C_1 + IS[p_1, \ldots]$, and the right side is $IS[0, p_2 - 1] + C_2 + IS[p_2, p_1 - 1] + C_1 + IS[p_1, \ldots]$;
   (b) $p_1 < p_2$, the left side is $IS[0, p_1 - 1] + C_1 + IS[p_1, p_2 - 1] + C_2 + IS[p_2, \ldots]$; and the right side is $IS[0, p_1 - 1] + C_1 + IS[p_1, p_2 - 1] + C_2 + IS[p_2, \ldots]$;
   (c) $p_1 = p_2$ and $Sid(IO_1) > Sid(IO_2)$, the left side is $IS[0, p_1 - 1] + C_2 + C_1 + IS[p_1, \ldots]$, and the right side is $IS[0, p_2 - 1] + C_2 + C_1 + IS[p_2, \ldots]$;
   (d) $p_1 = p_2$ and $Sid(IO_1) < Sid(IO_2)$, the left side is $IS[0, p_1 - 1] + C_1 + C_2 + IS[p_1, \ldots]$, and the right side is $IS[0, p_2 - 1] + C_1 + C_2 + IS[p_2, \ldots]$;

   in all cases, the left side equals the right side.
2) $IO_1 = I(p_1, c_1)$ and $IO_2 = M(p_2, c_2, v_2)$, the position parameters have three relations:
   (a) $p_1 > p_2$, the left side is $IS[0, p_2 - 1] + C_2 + IS[p_2 + 1, p_1 - 1] + C_1 + IS[p_1, \ldots]$, the right side is $IS[0, p_2 - 1] + C_2 + IS[p_2 + 1, p_1 - 1] + C_1 + IS[p_1, \ldots]$;
   (b) $p_1 = p_2$, the left side is $IS[0, p_1 - 1] + C_1 + C_2 + IS[p_1 + 1, \ldots]$, the right side is $IS[0, p_2 - 1] + C_1 + C_2 + IS[p_2 + 1, \ldots]$;
   (c) $p_1 < p_2$, the left side is $IS[0, p_1 - 1] + C_1 + IS[p_1, p_2 - 1] + C_2 + IS[p_2 + 1, \ldots]$, the right side is $IS[0, p_1 - 1] + C_1 + IS[p_1, p_2 - 1] + C_2 + IS[p_2 + 1, \ldots]$;

   in all cases, the left side equals the right side.
3) $IO_1 = M(p_1, c_1, v_1)$ and $IO_2 = M(p_2, c_2, v_2)$, as IO_1 and IO_2 have no impact on each other, the left side equals the right side, regardless of the position relations.

Combination 1, 2, and 3, the lemma is proved.

Lemma 2 Given a pair of intermediate-layer operations IO_1 and IO_2 generated on a state IS, if the effect of $IO_1 = M(p_1, c_1, v_1)$ is to mark the object C_1 in IS, the effect of $IO'_1 = T(IO_1, IO_2)$ is also to mark C_1 in the new state $IS \cdot IO_2$.

Proof Consider the operation type of IO_2,

1) if $IO_2 = M(p_2, c_2, v_2)$, as IO_2 does not move any object in IS, C_1 is still at position p_1 in $IS \cdot IO_2$; according to the transformation function, $Pos(IO'_1) = Pos(IO_1)$; thus the effect of IO'_1 is also to mark C_1 in $IS \cdot IO_2$;
2) if $IO_2 = I(p_2, c_2)$, the position parameters have 3 relations:
   (a) $p_1 > p_2$, $IS \cdot IO_2 = IS[0, p_2 - 1] + C_2 + IS[p_2, p_1 - 1] + C_1 + IS[p_1 + 1, \ldots]$, C_1 is at position $p_1 + 1$ in $IS \cdot IO_2$; according to the transformation function, $Pos(IO'_1) = Pos(IO_1) + 1$; thus the effect of IO'_1 is also to mark C_1 in $IS \cdot IO_2$;
   (b) $p_1 = p_2$, $IS \cdot IO_2 = IS[0, p_2 - 1] + C_2 + C_1 + IS[p_2 + 1, \ldots]$, C_1 is at position $p_2 + 1$ in $IS \cdot IO_2$; according to the transformation function, $Pos(IO'_1) = Pos(IO_1) + 1$; thus the effect of IO'_1 is also to mark C_1 in $IS \cdot IO_2$;
   (c) $p_1 < p_2$, $IS \cdot IO_2 = IS[0, p_1 - 1] + C_1 + IS[p_1 + 1, p_2 - 1] + C_2 + IS[p_2, \ldots]$, C_1 is at position p_1 in $IS \cdot IO_2$; according to the transformation function, $Pos(IO'_1) = Pos(IO_1)$; thus the effect of IO'_1 is also to mark C_1 in $IS \cdot IO_2$;

Combination 1 and 2, the lemma is proved.

According to Lemma 2, it is trivial to derive Lemma 3.

Lemma 3 Given an intermediate-layer mark operation IO, if the effect of IO is to mark the object C in IS, the effect of $IO' = LT(IO, L)$ is also to mark C in the state $IS \cdot L$.

Theorem 1 Given an undo command $Undo(O)$, ST-Undo would remove the effect of O but retain the effects of the other operations.

Proof We make the assumption that IO is the intermediate-layer operation corresponding to O, the current application-layer document state is S and intermediate-layer document state is IS, and IO takes effect on the object C at position p. And we introduce an auxiliary function F that returns a sequence of visible characters according to the given sequence of objects, e.g., $F(IS) = S$.

At the step 2, an intermediate-layer mark operation IO'_x is generated by the algorithm; according to Lemma 3, IO'_x takes effect on C. At the step 3, ST-Undo migrates IO'_x to an application-layer operation O_x. We have the following equa-

tions before and after $Undo(O)$ is handled,

$$S = F(IS[0, p-1]) + F(C) + F(IS[p+1, \ldots])$$
$$S \cdot O_x = F(IS[0, p-1]) + F(C') + F(IS[p+1, \ldots])$$

1) ST-Undo can preserve the effects of the other operations. According to the above equations, it can be observed that $Undo(O)$ has no impact on $IS[0, p-1]$ and $IS[p+1, \ldots]$, which record the effects of the other operations. Thus the effects of the other operations are retained after O is undone.
2) ST-Undo can remove the effect of O. Consider the operation type of O,
   (a) O is an insertion, if C is visible and C' is invisible, the effect of O_x is to delete $Char(C)$ from S; if C is invisible, C' must be invisible, because the effect of IO_x is to mark C invisible; and O_x would be an identity operation, which has no effect on S; in both cases, the effect of O is removed;
   (b) O is a deletion, if C is invisible and C' is visible, the effect of O_x is to insert $Char(C)$ between $F(IS[0, p-1])$ and $F(IS[p+1, \ldots])$; if C is invisible and C' is invisible, there exist other operations that have made C invisible, and O_x would be an identity operation; in both cases, the effect of O is removed.

Combination of 1 and 2, the theorem is proved.

By adapting the application-layer data and operation model to the intermediate-layer data and operation model, ST-Undo can separately handle do operations and undo operations. The consistency of intermediate-layer documents is guaranteed by the combination of a transformation function for intermediate-layer operations and an existing generic transformation control algorithm. As the undo algorithm never modifies the history buffer, the undo algorithm does not influence the do algorithm, which transforms do operations with the operations in the history buffer. The undo algorithm can trivially remove the effect of an operation and retain the effects of the other operations on the intermediate-layer documents. ST-Undo finally derives consistent shared and replicated documents from the consistent intermediate-layer documents by the operation migration scheme.

4 Operation migration scheme

We need an efficient data structure to represent intermediate-layer documents and support the conversion between editing operations to/from intermediate-layer operations. A data structure called hidden-object tree is designed to serve those purposes. We come up with this design according to several heuristics by our long-lasting observation and experimentation: (1) an intermediate-layer document essentially is an abstract linear data type, which has a variety of implementations, such as dynamic arrays, linked lists, and trees; (2) the visible objects in an intermediate-layer document are not necessary for operation migration, and therefore saving more objects increases space overhead but may not decrease time overhead; (3) in consideration of undo operations, invisible objects could be dynamically added or removed. This section will illustrate the structure of the hidden-object trees and the methods of performing the operation conversion by the hidden-object trees.

4.1 HOT: hidden-object tree

A hidden-object tree is a binary search tree that stores a set of invisible objects. Each node in the tree represents an invisible object and has five attributes,

- $Pos(node)$ represents the position attribute of a node,
- $Num(node)$ records the sum of mark effects performed on a node,
- $Size(node)$ records the number of nodes in the subtree rooted by a node,
- $Left(node)$ refers to the left child of a node,
- $Right(node)$ refers to the right child of a node.

Assume the root node of a subtree is located at position p ($p < n$), its left subtree represents the address space $[0, p-1]$ and the right subtree represents the address space $[p+1, n-1]$. More concretely, the position attribute of the left child equals its addressing position in the space $[0, n-1]$, while the right child is relatively addressed in the sense that its position is relative to its parent node.

We take an example to illustrate hidden-object trees. Initially, a shared document is "abcdef". After a sequence of operations $([D(3, d), D(2, c), D(3, f)])$ is executed, the document becomes "abe". Figure 5 depicts the tree. According to the size attribute of $node_d$, we can derive that there are three hidden objects in the intermediate-layer document: $node_d$, $node_c$, and $node_f$. According to the position attributes, we can derive their addressing positions: $Pos(node_d) = 3$, $Pos(node_c) = 2$, $Pos(node_d) + Pos(node_f) = 5$. And the numbers of hidden objects positioned before $node_d$, $node_c$, and $node_f$ are $Size(Left(node_d)) = 1$, $Size(Left(node_c)) = 0$, and $Size(Left(node_f)) + Size(Left(node_d)) + 1 = 2$.

A hidden-object tree allows fast lookup, addition, removal, and movement of hidden objects. To not be tedious, we introduce four basic routines that manipulate a hidden-object tree, without giving the implementations in detail,

- $Search(pos, HOT)$ retrieves the object whose addressing position is pos,
- $Add(pos, HOT)$ adds a new object with addressing position pos into HOT,
- $Remove(pos, HOT)$ removes the object whose addressing position is pos,
- $Shift(pos, HOT)$ increments the position attribute of the objects by one whose addressing position is greater than or equal to pos.

4.2 Operation migration

The operation migration scheme calls *CountLR* and *CountRL* to count the number of hidden objects before a given position in two directions. In the following descriptions, if a node is null, its size attribute value is treated as 0 in default.

Given a position index p in the application-layer document, the function *CountLR* in Algorithm 2 traverses the hidden-object tree to get the number of hidden objects m positioned

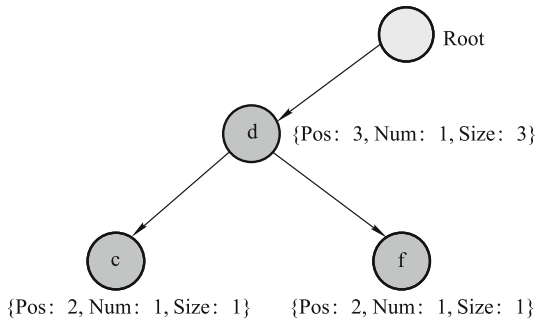


Fig. 5 A hidden-object tree

Algorithm 2 CountLR(p,HOT)

Input: p is a position in S;
Output: m is the number of objects before $IS[p+m]$;
1: $m \leftarrow 0$; if ($p < 0$) return 0;
2: $cnode \leftarrow Left(Root(HOT))$;
3: **While** $cnode \neq null$ **do**
4: $lnode \leftarrow Left(cnode)$;
5: **if** $p + Size(lnode) < Pos(cnode)$ **then**
6: $cnode \leftarrow lnode$;
7: **else**
8: $p \leftarrow p + Size(lnode) + 1 - Pos(cnode)$;
9: $m \leftarrow m + Size(lnode) + 1$;
10: $cnode \leftarrow Right(cnode)$;
11: **end if**
12: **end while**
13: Return m;

before the pth visible object (the target object) in the intermediate-layer document. For a traversed non-null node $cnode$, if $p + Size(lnode) < Pos(cnode)$, the target object must be positioned before $cnode$ and the search continues in the subtree $Left(cnode)$; otherwise, all the objects in $Left(cnode)$ and $cnode$ itself are positioned before the target object, the search continues in the subtree $Right(cnode)$.

Given a position index p in the intermediate-layer document, the function *CountRL* in Algorithm 3 traverses the hidden-object tree to get the number of hidden objects m positioned before the target object whose addressing position is at position p in the intermediate-layer document. For a traversed non-null node $cnode$, if $p < Pos(cnode)$, the target object must be positioned before $cnode$ and the search continues in the subtree $Left(cnode)$; if $p = Pos(cnode)$, the operation takes effect on $cnode$ and the search ends up in advance; if $p > Pos(cnode)$, all the objects in the subtree $Left(cnode)$ and $cnode$ itself are positioned before the target object, the search continues in the subtree $Right(cnode)$.

The function MAI in Algorithm 4 migrates an application-layer operation O to an intermediate-layer operation IO. The position mapping is computed by *CountLR*. There are two cases. When O is $I(p,c)$, c is mapped to the object that is located immediately after the $(p-1)$th visible object, the function calls $CountLR(p-1)$ and returns an intermediate-layer operation $I(p+m,c)$. When O is $D(p,c)$, c is mapped to the pth visible object, and the function calls $CountLR(p)$ and returns an intermediate-layer operation $M(p+m,c,+1)$.

The function *MIA* in Algorithm 5 migrates an intermediate-

Algorithm 3 CountRL(p,HOT)

Input: p is a position in IS;
Output: m is the number of hidden objects in $IS[p]$;
1: $m \leftarrow 0$; if ($p = 0$) return 0;
2: $cnode \leftarrow Left(Root(HOT))$;
3: **while** $cnode \neq null$ **do**
4: $lnode \leftarrow Left(cnode)$;
5: **if** $p < Pos(cnode)$ **then**
6: $cnode \leftarrow lnode$;
7: **els if** $p = Pos(cnode)$ **then**
8: $m \leftarrow m + Size(lnode)$; break;
9: **else**
10: $p \leftarrow p - Pos(cnode)$;
11: $m \leftarrow m + Size(lnode) + 1$;
12: $cnode \leftarrow Right(cnode)$;
13: **end if**
14: **end while**
15: Return m;

Algorithm 4 MAI(O,HOT)

Input: O is an application-layer operation
Output: IO is an operation generated on HOT
1: **if** O is $I(p,c)$ **then**
2: $m \leftarrow CountLR(p-1, HOT)$;
3: $IO \leftarrow I(p+m,c)$;
4: **else**
5: $m \leftarrow CountLR(p, HOT)$;
6: $IO \leftarrow M(p+m,c,+1)$;
7: **end if**
8: Return IO;

Algorithm 5 MIA(IO,HOT)

Input: IO is an operation generated on HOT
Output: O is an application-layer operation
1: **if** IO is $I(p,c)$ **then**
2: $m \leftarrow CountRL(p, HOT)$;
3: $O \leftarrow I(p-m,c)$;
4: **else**
5: $node \leftarrow Search(p, HOT)$;
6: **if** IO is $M(p,c,-1)$ and $node = null$ **then**
7: $m \leftarrow CountRL(p, HOT)$;
8: $O \leftarrow I(p-m,c)$;
9: **els if** IO is $M(p,c,+1)$ and $Num(Node) = 1$ **then**
10: $m \leftarrow CountRL(p, HOT)$;
11: $O \leftarrow D(p-m,c)$;
12: **else**
13: $O \leftarrow identity$;
14: **end if**
15: **end if**
16: Return O;

layer operation IO to an application-layer operation O after the effect of IO has been recored in HOT. The position mapping is computed by *CountRL*. There are three cases,

1) when IO is $I(p,c)$, IO is a normal do operation that intends to insert a visible object at position p, and the function returns an application-layer operation $I(p-m,c)$;
2) when IO is $M(p,c,-1)$, IO is an undo operation that intends to make the object $node$ at position p visible; if $node$

does not exist in HOT, IO has made *node* visible, and the function returns an application-layer operation $I(p-m, c)$; otherwise, *node* is invisible and the function returns an *identity* operation;

3) when IO is $M(p, c, +1)$, IO could a do or undo operation that intends to make the visible object *node* at position p invisible; if $Num(node) = 1$, IO has made *node* invisible, and the function returns an application-layer operation $D(p - m, c)$; otherwise, *node* was invisible before executing IO and the function returns an *identity* operation.

4.3 Operation effect recording

An intermediate-layer operation could be generated in handling do and undo operations. The effects of intermediate-layer operations are recorded in the hidden-object tree to stay consistent with the application-layer document.

The Algorithm 6 describes how to apply an intermediate-layer operation IO on a hidden-object tree HOT. In line 2, when IO is an insert operation, the hidden objects whose addressing position is not less than $Pos(IO)$ are shifted to the right by one position. In lines 4–12, when IO is a mark operation, the algorithm first finds the object whose addressing position is $Pos(IO)$; if no such object, a new visible object will be created and added into HOT; if the object exists, its visible attribute is updated and the object will be removed if it becomes a visible object.

The operation migration scheme efficiently stores the intermediate-layer documents and manages the conversion between intermediate-layer and application-layer operations. As an intermediate-layer document only stores invisible objects, both space and time overhead of the scheme are dependent on the number of deleted characters rather than the full set of characters that ever appear in a shared and replicated document.

5 A working example

In this section, we use a representative co-editing scenario to explain the proposed algorithm step by step in handling do and undo operations and discuss the practical usefulness of ST-Undo in real-world complex co-editing systems in detail.

Algorithm 6 Update(IO, HOT)

Input: IO is an intermediate-layer operation generated on HOT
Output:

```
1: if IO is I(p, c) then
2:    Shift(p, HOT);
3: else
4:    node ← Search(p, HOT);
5:    if node ≠ null
6:       Num(node) ← Num(node) + v;
7:       if Num(node) = 0 then
8:          Remove(p, HOT);
9:       end if
10:   else
11:      Add(p, HOT);
12:   end if
13: end if
```

In Fig. 6, three users collaboratively edit a shared text document. Initially, the document state is "a". Users concurrently generate do and undo operations: User A at site 1 generates O_1 to delete the character "a", User B at site 2 generates O_2 to delete the character "a" and then issues an undo action $Undo(O_2)$, and User C at site 3 generates O_3 to insert a character "x" before "a". We elaborate the collaboration processes by ST-Undo as follows.

At site 1, S_{10}="a", $H_{10} = []$, $tree_{10} = \{\}$;

1) User A generates $O_1 = D(0, a)$, $S_{11} =$" ",
   (a) migrate O_1 to $IO_1 : M(0, a, +1)$;
   (b) add IO_1 to the history buffer, $H_{11} = [IO_1]$;
   (c) update $tree_{10}$ by IO_1, $tree_{11} = \{\{'a', 0, 1, 1\}\}$[2];
2) Receive $IO_2 = M(0, a, +1)$,
   (a) transform IO_2 by $LT(IO_2, [IO_1])$, $IO_2' = M(0, a, +1)$;
   (b) add IO_2 to the history buffer, $H_{12} = [IO_1, IO_2']$;
   (c) migrate IO_2' to O_2' :*identity*;
   (d) update $tree_{11}$ by IO_2', $tree_{12} = \{\{'a', 0, 2, 1\}\}$;
   (e) apply O_2', $S_{12} = S_{11} \cdot identity =$" ";
3) Receive $Undo(IO_2)$,
   (a) transform IO_x by $LT(IO_x, [])$, $IO_x' = M(0, a, -1)$;
   (b) migrate IO_x' to O_x :*identity*;
   (c) update $tree_{12}$ by IO_x', $tree_{13} = \{\{'a', 0, 1, 1\}\}$;
   (d) apply O_x, $S_{13} = S_{12} \cdot identity =$" ".
4) Receive $IO_3 = I(0, x)$,
   (a) transform IO_3 by $LT(IO_3, [IO_1, IO_2'])$, $IO_3' = I(0, x)$;
   (b) add IO_3 to the history buffer, $H_{14} = [IO_1, IO_2', IO_3']$;
   (c) migrate IO_3' to $O_3' : I(0, x)$;
   (d) update $tree_{13}$ by IO_3', $tree_{14} = \{\{'a', 1, 1, 1\}\}$;
   (e) apply O_3', $S_{14} = S_{13} \cdot I(0, x) =$"x";

At site 2, S_{20}="a", $H_{20} = []$, $tree_{20} = \{\}$;

1) User B generates $O_2 = D(0, a)$, $S_{21} =$" ";
   (a) migrate O_2 to $IO_2 : M(0, a, +1)$;
   (b) add IO_2 to the history buffer, $H_{21} = [IO_2]$;
   (c) update $tree_{20}$ by IO_2, $tree_{20} = \{\{'a', 0, 1, 1\}\}$;
2) User B issues $Undo(IO_2)$,
   (a) transform IO_x by $LT(IO_x, [])$, $IO_x' = M(0, a, -1)$;
   (b) migrate IO_x' to $O_x : I(0, a)$;
   (c) update $tree_{21}$ by IO_x', $tree_{22} = \{\}$;
   (d) apply O_x, $S_{22} = S_{21} \cdot I(0, a) =$"a".
3) Receive $IO_1 = M(0, a, +1)$,
   (a) transform IO_1 by $LT(IO_1, [IO_2])$, $IO_1' = M(0, a, +1)$;
   (b) add IO_1 to the history buffer, $H_{23} = [IO_1, IO_2']$, where$IO_2' = T(IO_2, IO_1)$;
   (c) migrate IO_1' to $O_1' : D(0, a)$;

[2] For clarity, a node is denoted by $\{Char, Pos, Num, Size\}$

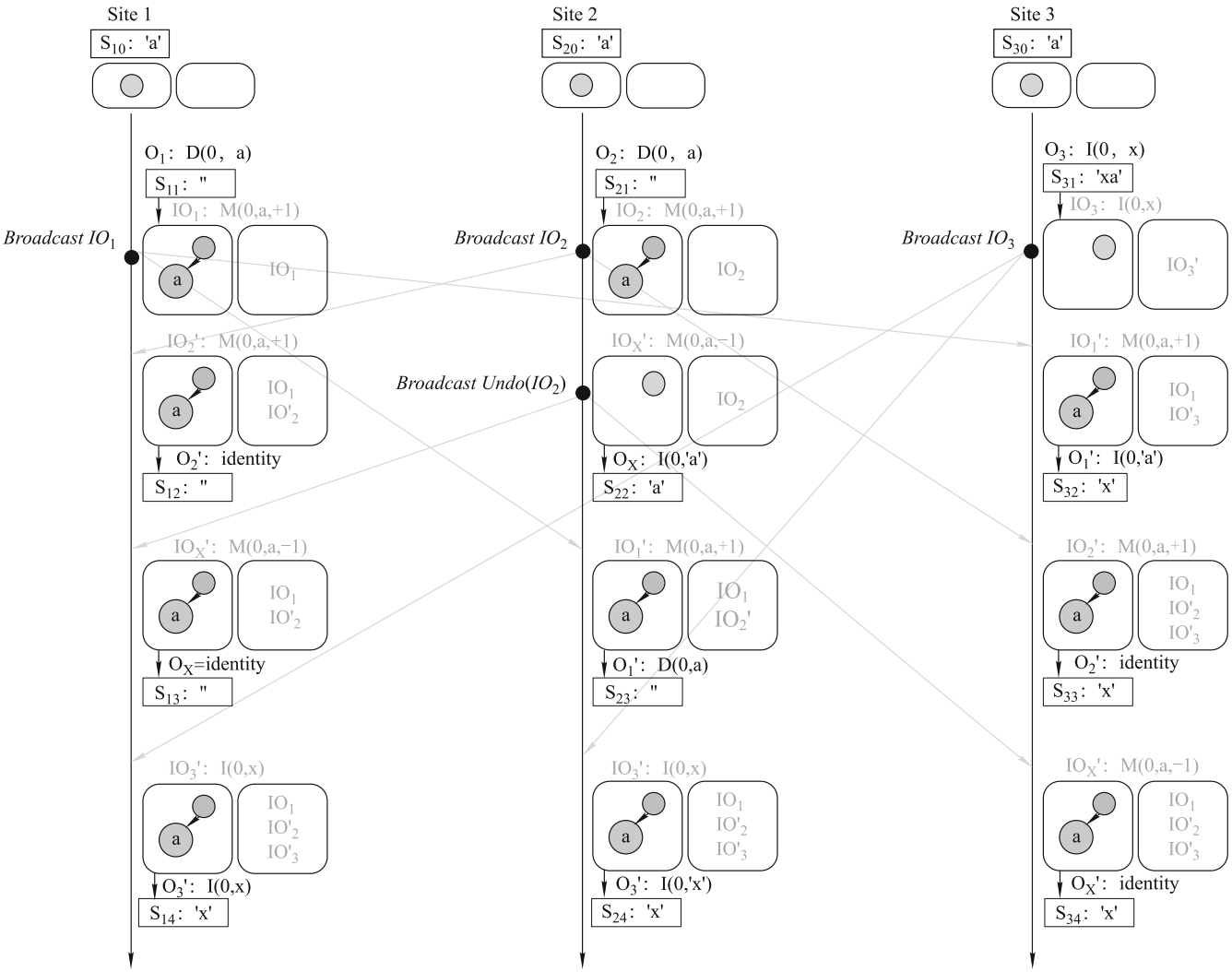


Fig. 6 A working example

(d) update $tree_{22}$ by $IO_1{}'$, $tree_{23} = \{\{'a', 0, 1, 1\}\}$;
(e) apply O'_1, $S_{23} = S_{22} \cdot D(0, a) =$" ";

4) Receive $IO_3 = I(0, x)$,
(a) transform IO_3 by $LT(IO_3, [IO_1, IO_2{}'])$, $IO_3{}' = I(0, x)$;
(b) add IO_3 to the history buffer, $H_{24} = [IO_1, IO_2{}', IO_3{}']$;
(c) migrate $IO_3{}'$ to $O'_3 : I(0, x)$;
(d) update $tree_{23}$ by $IO_3{}'$, $tree_{24} = \{\{'a', 1, 1, 1\}\}$;
(e) apply O'_3, $S_{24} = S_{23} \cdot I(0, x) =$"x";

At site 3, S_{30}="a", $H_{30} = []$, $tree_{30} = \{\}$;

1) User C generates $O_3 = I(0, x)$, $S_{31} =$"xa";
(a) migrate O_3 to $IO_3 : I(0, x)$;
(b) add IO_3 to the history buffer, $H_{31} = [IO_3]$;
(c) update $tree_{30}$ by IO_3, $tree_{31} = \{\}$;

2) Receive $IO_1 = M(0, a, +1)$,
(a) transform IO_1 by $LT(IO_1, [IO_3])$, $IO_1{}' = M(1, a, +1)$;
(b) add IO_1 to the history buffer, $H_{32} = [IO_1, IO_3{}']$, where $IO'_3 = T(IO_3, IO_1)$;
(c) migrate $IO_1{}'$ to $O'_1 : D(1, a)$;
(d) update $tree_{31}$ by $IO_1{}'$, $tree_{32} = \{\{'a', 1, 1, 1\}\}$;
(e) apply O'_1, $S_{32} = S_{31} \cdot D(1, a) =$"x";

3) Receive $IO_2 = M(0, a, +1)$,
(a) transform IO_2 by $LT(IO_2, [IO_1, IO_3{}'])$, $IO_2{}' = M(1, a, +1)$;
(b) add IO_2 to the history buffer, $H_{33} = [IO_1, IO_2{}', IO_3{}']$, where $IO_2{}' = T(IO_2, IO_1)$ and $IO_3{}' = LT(IO_3, [IO_1, IO'_2])$;
(c) migrate $IO_2{}'$ to O'_2 :*identity*;
(d) update $tree_{32}$ by $IO_2{}'$, $tree_{33} = \{\{'a', 1, 2, 1\}\}$;
(e) apply O'_2, $S_{33} = S_{32} \cdot identity =$"x";

4) Receive $Undo(IO_2)$,
(a) transform $IO_x : M(0, a, -1)$ by $LT(IO_x, [IO_3{}'])$, $IO_x{}' = M(1, a, -1)$;
(b) migrate $IO_x{}'$ to O_x :*identity*;
(c) update $tree_{33}$ by $IO_x{}'$, $tree_{34} = \{\{'a', 1, 1, 1\}\}$;

(d) apply O_x, $S_{34} = S_{33} \cdot identity$ ="x".

From the above example, it can be observed that the replicated documents at different cooperating sites finally are consistent after executing the same set of do and undo operations, regardless of the execution orders. Furthermore, the correct undo effect is delivered to the users: undoing an operation (e.g., O_2) only removes its own effect but preserves the effects of other operations. The additional metadata only maintains deleted characters not the full set of characters.

ST-Undo is applicable to the co-editing systems of which the editing documents are linearly addressed. In other words, if a single-user editor allows the objects in the documents to be accessed by positional index and maintains the objects in a linear-addressing space, the single-user editor can be extended to a multi-user collaborative editors by ST-Undo. Text editors usually meets the linear-addressing requirement.

To apply ST-Undo in a real-world complex co-editing system, we need to devise a functional component capable of translating the user actions that can change the document states to/from the character-wise operations that can be handled by ST-Undo. If the user-level operations are converted to the algorithm-level operations, complicated collaboration scenarios from the perspective of users are simplified by the scenarios consisting of the algorithm-level operations with causal and concurrency relations. In the above example, for the convenience of discussion, we assume that users directly generate the operations handled by ST-Undo. However, in real-world co-editing systems, the user-level operations are far more complex than the two character-wise operations. Translation of the complex user-level operations to the simple algorithm-level operations is a challenging task. And it is beyond the scope of this paper.

6 Comparison to prior approaches

UNO [25, 26] and ABTU [28] are two relatively new undo approaches built on the OT-based consistency maintenance algorithms capable of solving the abnormal ordering issue. UNO [25, 26] and the CRDT-based undo solutions [46–48] have the same usage of object sequences. And the time/space efficiency of UNO and the CRDT-based undo solutions both are dominated by the object sequences. We consider UNO to be a representative of object-sequence centric undo approaches. As UNO and ABTU have been compared with the prior approaches in the literature [25, 26, 28, 30], the following discussions focus on the comparison of ST-Undo, UNO and ABTU by theoretical analysis and experimental evaluation.

6.1 Theoretical analysis

6.1.1 Comparison of ST-Undo and UNO

UNO (undo as a new operation) [25, 26] extends selective undo functionality on TTF systems [36]. At each cooperating site, TTF systems maintain an object sequence consisting of visible and invisible objects in a dynamic array. Each object corresponds to a character ever appearing in the shared documents. A special transformation function is used in TTF systems to transform the operations that manipulate the object sequence. To do the conversion between editing operations and object-sequence operations, TTF systems compute the position conversion by traversing the object sequence and counting the visible and invisible objects.

UNO defines a new operation as the inverse of delete operations. Moreover, the TTF transformation function is extended to support this new operation. After users issue an undo command, UNO interprets the undo command as a normal do operation and sends the do operation, instead of the undo command, to other cooperating sites. Consequently, UNO handles an undo command generated at remote sites like a remote do operation.

ST-Undo has two distinctive differences with UNO. First, UNO stores both do and undo operations in the history buffer, whereas ST-Undo only stores normal do operations in the history buffer. This difference brings multiple benefits to ST-Undo: (1) the algorithm becomes simple, because the algorithm does not need to consider the interference between do and undo, (2) ST-Undo has better efficiency than UNO in doing transformations. Second, UNO maintains all the visible and invisible objects in the object sequence, but ST-Undo only maintains a set of invisible objects stored in a binary search tree. Moreover, UNO never removes objects from the object sequence, whereas ST-Undo removes an object in the tree when it becomes visible. With these differences, ST-Undo can outperform UNO in both time and space efficiency.

6.1.2 Comparison of ST-Undo and ABTU

ABTU (ABT-based undo) [28] is established on the framework of ABT [39], which aims to preserve object order relations. ABT arranges the history buffer as insert-before-delete, i.e., insertions are placed before deletions. In local operation processing, given a user-generated operation O, ABT transforms O with deletion part to get a transformed operation O' and broadcast O' to other cooperating sites. In remote operation processing, given a received operation O, ABT transforms O with its concurrent operations in the insertion part to get an intermediate operation O'', and then transforms O'' with the deletion part to get the final result O'. In handling both local and remote operations, ABT updates the history buffer to keep the insert-before-delete arrangement.

To improve the time efficiency of ABT, ABTU stores all the operations in the history buffer with respect to the effect position of operations, which is mandated as a total order. To be capable of handling undo operations, ABTU associates every operation with five vector timestamps to model the various relations between the operations defined in ABTU. Compared to ABTU, ABT associates every operation with one vector timestamp to detect causal and concurrent relations.

ST-Undo has three distinctive differences with ABTU. First, in processing a normal do operation, the transformation cost of ABTU depends on all the operations in the history buffer, whereas the transformation cost of ST-Undo depends on its concurrent operations. Second, ABTU models complex dependency and do/undo/redo relations among operations by vector timestamps, whereas ST-Undo only models causal dependency between a normal do operation and its undo command, which is compatible with general causal relations. Third, ABTU does not fully respect the undo effect specified in Definition 3, but ST-Undo does fully respect that undo effect. For example, if O_1 inserts a character and later O_2 deletes the character, then

ABTU considers that O_2 is dependent on O_1. Unless O_1 has been undone, $Undo(O_2)$ is not permitted. In the case that multiple concurrent operations delete the same character, undoing one of them would re-insert the character in the document. Consequently, in ABTU, undoing an operation may remove not only its own effect but also the other operations' effects.

6.1.3 Comprehensive comparison

In terms of metadata, ST-Undo maintains a history buffer and a hidden-object tree, ABT maintains a history buffer, and UNO maintains a history buffer and an object sequence. The time and space costs are related to these metadata. It is worth pointing out that the history buffer of ST-Undo only stores normal do operations, whereas the history buffers of UNO and ABT store both do and undo operations. Moreover, ST-Undo and UNO use scalar timestamps, but ABTU uses vector timestamps. To evaluate the performance, we consider the following variables,

- n denotes the number of cooperating sites,
- N denotes the number of characters ever appearing in shared documents, and N_d denotes the number of deleted characters;
- M denotes the number of all operations accumulated in the history buffer, M_c denotes the number of operations that are concurrent to a given operation in the history buffer, M_d denoted the number of normal do operations accumulated in the history buffer.

For UNO and ST-Undo, an object in the object sequence, a node in the hidden object tree, and an operation in the history buffer, have a fixed size. Thus the space complexity of UNO and ST-Undo can be expressed as $O(N + M)$ and $O(N_d + M_d)$, respectively. For ABTU, an operation in the history buffer is associated with five vector timestamps, and the size of a vector timestamp is proportional to the number of cooperating sites. The space complexity of ABTU is expressed by $O(M \times n)$. Table 1 summarizes the space complexity.

The time cost of UNO is dominated by operation transformation and position conversion. Both the worst-time complexity and average-time complexity of the position conversion procedure are $O(N)$. In local do processing, the main computation is the position conversion, and the worst-case and average-case time complexity are $O(N)$. In local undo processing, the main computation includes the operation transformation and the position conversion. In the worst case, an operation is transformed with all the operations in the history buffer, and the time cost is $O(M)$. In the average case, assume every operation has the same possibility being undone, the number of transformations is $\sum_{k=1}^{M} k \times \frac{1}{M} = \frac{M}{2}$. The worst-case and average-case time complexity are $O(M+N)$. UNO handles remote do and undo operations in the same way, which first transforms a given operation with its concurrent operations and then calls the position conversion. Thus the worst-case and average-case time complexity of UNO in remote do/undo processing are $O(M_c + N)$.

Table 1 Space complexity of UNO, ABTU, and ST-Undo

UNO	ABTU	ST-Undo
$O(N + M)$	$O(M \times n)$	$O(N_d + M_d)$

No matter what operations are handled, the time cost of ABTU includes two parts: operation placement and operation transformation. The operation placement is similar to a step of insertion sort. Its worst-case and average-case time complexity are $O(M)$. In local do processing, the operation placement costs $O(1)$ and the operation transformation costs $O(M)$, and the worst-case and average-case time complexity are $O(M)$. In remote do processing, the operation placement and the operation transformation both cost $O(M)$, and the worst-case and average-case time complexity are $O(M)$. In local undo processing, the operation placement costs $O(1)$ and the operation transformation costs $O(M)$, and the worst-case and average-case time complexity are $O(M)$. In remote undo processing, the operation placement and the operation transformation both cost $O(M)$, and the worst-case and average-case time complexity are $O(M)$.

The time cost of ST-Undo is dominated by operation transformation and operation migration. The worst-case and average-case time complexity of the operation migration procedure are $O(N_d)$ and $O(\log(N_d))$ respectively. In local do processing, the main computation is the operation migration, and the worst-case and average-case time complexity are $O(N_d)$ and $O(\log(N_d))$ respectively. In remote do processing, ST-Undo first transforms a given operation against its concurrent operations in the history buffer and then calls the operation migration procedure, and the worst-case and average-case time complexity are $O(M_c + N_d)$ and $O(M_c + \log(N_d))$ respectively. ST-Undo handles local/remote undo commands in the same way, which first calls the operation transformation and then operation migration. In the worst case, an operation is transformed with all the operations in the history buffer; in the average case, assume every operation has the same possibility being undone, the number of transformations is $\sum_{k=1}^{M_d} k \times \frac{1}{M_d} = \frac{M_d}{2}$. The worst-case and average-case time complexity are $O(M_d + N_d)$ and $O(M_d + \log(N_d))$.

Table 2 summarizes the time complexity. The relations among the variables in big O notation generally can be expressed as: $N > M > N_d$, $M > M_d$, and $M \gg M_c$. It can be seen that ST-Undo outperforms the other two algorithms in both time and space efficiency.

6.2 Performance evaluation

We conducted two experiments to measure the time costs of three algorithms in handling do and undo operations. All the algorithms were implemented in Java and executed on a computer running Windows 10 Education with an Intel CPU Core i7-6600U (2.6 GHz) and 4 GB RAM. The two experiments follow the same design pattern as the prior performance studies

Table 2 Time complexity of UNO, ABTU, and ST-Undo

	Do/Undo	UNO	ABTU	ST-Undo
Worst case	Local Do	$O(N)$	$O(M)$	$O(N_d)$
	Remote Do	$O(M_c + N)$		$O(M_c + N_d)$
	Local Undo	$O(M + N)$	$O(M)$	$O(M_d + N_d)$
	Remote Undo	$O(M_c + N)$		
Average case	Local Do	$O(N)$	$O(log(N_d))$ $O(M)$	
	Remote Do	$O(M_c + N)$		$O(M_c + log(N_d))$
	Local Undo	$O(M + N)$	$O(M)$	$O(M_d + log(N_d))$
	Remote Undo	$O(M_c + N)$		

of co-editing algorithms [41, 49]. Each experiment is run 20 times to average the measured times.

In the first experiment, we test the performance of handling normal do operations. The collaboration workloads are simulated by the following processes: (1) site 1 generates n_1 editing operations and concurrently site 2 generates n_2 editing operations, (2) the n_1 operations generated by site 1 are received and handled at site 2. The n_1 operations from site 1 are concurrent to the n_2 operations from site 2. Now the history buffer of site 1 stores n_1 operations and the history buffer of site 2 stores n_1+n_2 operations. At the first step, we measure the time cost of handling a newly generated operation at site 1. At the second step, we measure the time cost of handling a remote operation that is generated at site 1 and executed at site 2. In this experiment, we vary n_1 from 200 to 1000 with step 200 and set n_2 to a constant value 1000, and the shared document initially contains 1000 characters.

The results of the first experiment are shown in Figs. 7 and 8. In the measurement of local/remote do operations, the time cost of UNO and ABTU both grow linearly, whereas the time cost of ST-Undo has no significant change, with the increasing number of do operations in the history buffer. More concretely, in local operation processing, when the history buffer accumulates 1000 operations, UNO spends about 80us more than ST-Undo, and ABTU spends about 50us more than ST-Undo; in remote operation processing, when the history buffer accumulates 2000 operations (1000 concurrent operations and 1000 causally-related operations), UNO and ABTU spend approximately 130us and 50us more than ST-Undo.

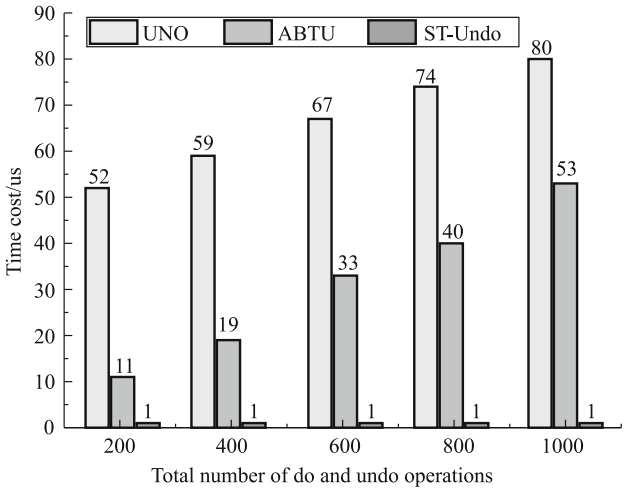


Fig. 7 Time cost in handling a local do operation

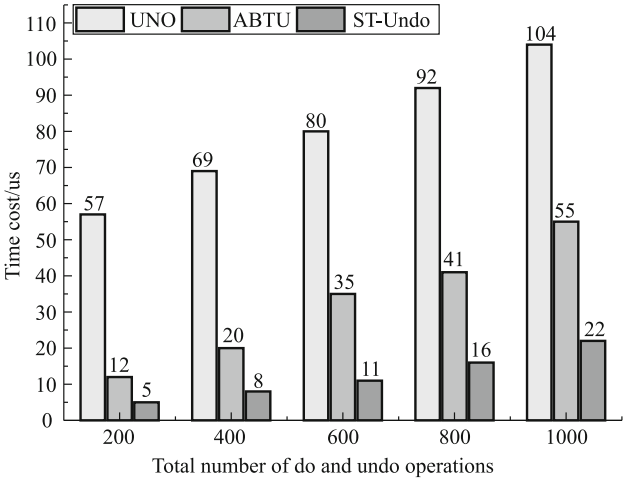


Fig. 8 Time cost in handling a remote do operation

In the second experiment, we test the performance of handling undo operations. The collaboration workloads are simulated by the following processes: (1) site 1 generate n_1 editing operations and then undoes the n_1 operations, and site 2 concurrently generates n_2 editing operations and then undoes the n_2 editing operations; (2) site 2 receives the n_1 editing operations and n_1 undo operations from site 1. The $2n_1$ do/undo operations from site 1 are concurrent to the $2n_2$ do/undo operations from site 2. For UNO and ABTU, the history buffer at site 1 contains $2n_1$ operations and the history buffer at site 2 contains $2n_2$ operations; for ST-Undo, the history buffer at site 1 and site 2 only contain n_1 and n_2 normal do operations respectively. At the first step, we measure the time cost in handling a local undo operation at site 1. At the second step, we measure the time cost in handling a remote undo operation that is generated at site 1 and executed at site 2. In this experiment, we vary n_1 from 100 to 500 with step 100 and set n_2 to a constant value 500, and the shared document initially contains 1000 characters.

The results of the second experiment are shown in Figs. 9 and 10. In handling undo operations, the time cost of the three algorithms have a linear growth with the increasing number of do and undo operations. In specific, in handling local undo operations, the execution time of UNO is longer than that of ST-Undo roughly by a factor of six, and the execution time of ABTU is about two times longer than that of ST-Undo; in handling remote undo operations, ST-Undo and UNO have a close growth rate with the increasing number of do and undo operations, which is lower than that of ABTU.

From the above experiments, it can be observed that UNO and ABTU extend the undo capability at the price of degrading the performance of normal do operations. Compared to UNO and ABTU, ST-Undo can retain the high-efficiency of consistency maintenance for do operations while providing an efficient selective undo capability. The experiments confirmed that ST-Undo is more efficient than UNO and ABTU.

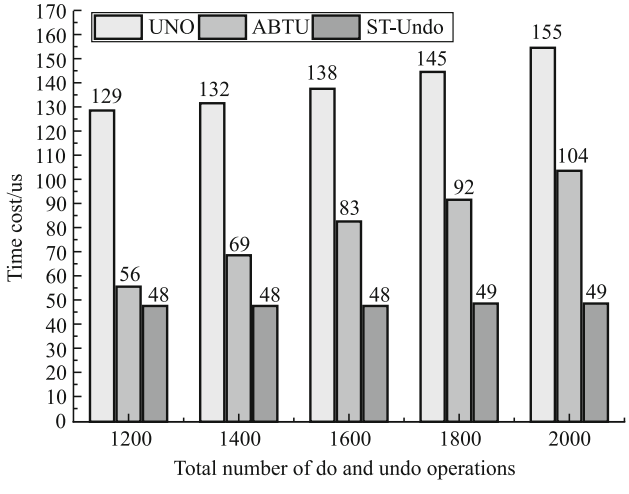


Fig. 9 Time cost in handling a local undo operation

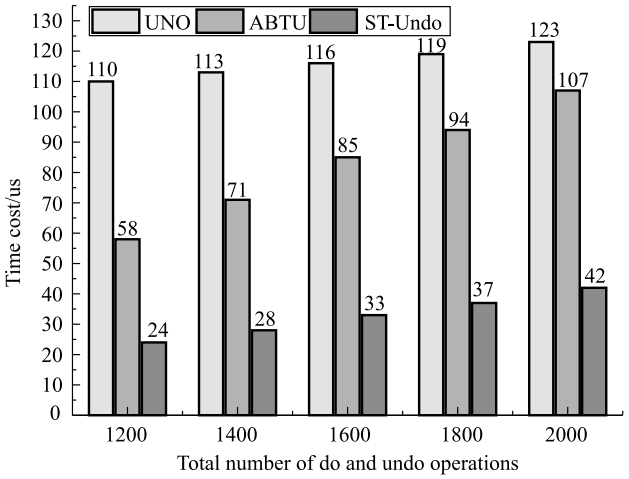


Fig. 10 Time cost in handling a remote undo operation

7 Conclusion

Undo is an essential functionality for interactive applications. In computer supported collaboration environments, enabling multiple users undo any operation at any time is a great technical challenge. In this work, we present a semi-transparent undo approach for multi-user collaborative editors. Our idea is to separate the undo-related problems from the do-related problems and avoid the interference between do and undo operations as much as possible. Main innovations in our approach include: a multi-layer OT framework capable of handling both do and undo operations in a unified way, an undo algorithm that transparently works with existing do algorithms, an efficient operation migration scheme that has logarithmic time complexity and linear space complexity in the number of deleted objects. Compared to the prior approaches, the proposed algorithm not only can deliver the desired undo effect to collaborative users but also has better efficiency in time and space overhead.

Collaborative systems are becoming important computer-aided tools for human-to-human collaboration. In future research, we continue our work in the following directions. Firstly, we will extend our idea to mobile and cloud computing environments [50]. Secondly, we will enhance our algorithm in real-world applications, such as collaborative digital design [51–61] and knowledge-sharing social network or community [62–66]. Thirdly, we will discuss how to accelerate massive-scale collaborative applications with parallel and optimization methods [67–72].

Acknowledgements This work was supported by National Key R&D Program of China (2017YFB0503004), the National Natural Science Foundation of China (Grant No. 62072348), China Postdoctoral Science Foundation (2019M662709), and Natural Science Foundation of Hubei Province (2016FC0106305 and 2019CFB627).

References

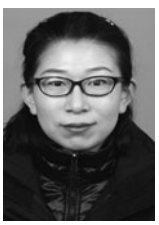
1. Cho B, Sun C Z, Ng A. Issues and experiences in building heterogeneous co-editing systems. Proceedings of the ACM on Human-Computer Interaction, 2019, 3(GROUP): 1–28
2. Fan H F, Li K, Li X Z, Song T Y, Zhang W Z, Shi Y. CoVSCode: a novel real-time collaborative programming environment for lightweight IDE. Applied Sciences, 2019, 9(21): 4642
3. Mirri S, Roccetti M, Salomoni P. Collaborative design of software applications: the role of users. Human-centric Computing and Information Sciences, 2018, 8(1): 1–20
4. Liang Y Q, He F Z, Zeng X T. 3D mesh simplification with feature preservation based on whale optimization algorithm and differential evolution. Integrated Computer-Aided Engineering, 2020, 27(4): 417–435
5. Touhafi A, Braeken A, Tahiri A, Zbakh M. Coderlabs: a cloud-based platform for real-time online labs with user collaboration. Concurrency and Computation: Practice and Experience, 2018, 30(12): e4377
6. Gao L P, Gao D F, Xiong N X, Lee C. CoWebDraw: a real-time collaborative graphical editing system supporting multi-clients based on HTML5. Multimedia Tools and Applications, 2018, 77(4): 5067–5082
7. Ignat C L, André L, Oster G. Enhancing rich content wikis with real-time collaboration. Concurrency and Computation: Practice and Experience, 2021, 33(8): e4110
8. Fan H F, Zhu H M, Liu Q, Shi Y, Sun C Z. A novel DAL scheme with shared-locking for semantic conflict prevention in unconstrained real-time collaborative programming. IEEE Access, 2017, 5: 22566–22583
9. Ng A, Sun C Z. Operational transformation for real-time synchronization of shared workspace in cloud storage. In: Proceedings of ACM International Conference on Supporting Group Work. 2016, 61–70
10. Junuzovic S, Dewan P. Towards self-optimizing collaborative systems. In: Proceedings of ACM Conference on Computer Supported Cooperative Work. 2012, 1421–1430
11. Bartel J W, Dewan P. Towards multi-domain collaborative toolkits. In: Proceedings of ACM Conference on Computer Supported Cooperative Work. 2012, 1297–1306
12. Li G, Zhu H Y, Lu T, Ding X H, Gu N. Is it good to be like wikipedia?: exploring the trade-offs of introducing collaborative editing model to Q&A sites. In: Proceedings of ACM Conference on Computer Supported Cooperative Work and Social Computing. 2015, 1080–1091
13. Olson J S, Wang D K, Olson G M, Zhang J W. How people write together now: beginning the investigation with advanced undergraduates in a project course. ACM Transactions on Computer-Human Interaction, 2017, 24(1): 1–40
14. Wang D K, Tan H D, Lu T. Why users do not want to write together when they are writing together: users' rationales for today's collaborative writing practices. Proceedings of the ACM on Human-Computer Interaction, 2017, 1(CSCW): 1–18
15. Cai W W, Ng A, Sun C Z. Some discoveries from a concurrency benchmark study of major cloud storage systems. In: Proceedings of International Conferenceon on Cooperative Design, Visualization, and Engineering. 2018, 44–48
16. Sun C Z, Chen D, Jia X H, Zhang Y C, Yang Y. Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems. ACM Transactions on Computer-Human Interaction, 1998, 5(1): 63–108
17. Ng A, Liu F, Xia S, Shen H F, Sun C Z. CoMaya: incorporating advanced collaboration capabilities into 3D digital media design tools. In: Proceedings of ACM Conference on Computer Supported Cooperative Work. 2008, 5–8
18. Sun C Z, Xia S, Sun D, Chen D, Shen H F, Cai W T. Transparent adaptation of single-user applications for multi-user real-time collaboration. ACM Transactions on Computer-Human Interaction, 2006, 13(4): 531–582
19. Xu B M, Gao Q, Li C Y. Reusing single-user applications to create collaborative multi-member applications. Advances in Engineering Software, 2009, 40(8): 618–622
20. Fan H F, Sun C Z, Shen H F. ATCoPE: any-time collaborative programming environment for seamless integration of real-time and non-real-time teamwork in software development. In: Proceedings of ACM Interna-

tional Conference on Supporting Group Work. 2012, 107–116
21. Prakash A, Knister M J. A framework for undoing actions in collaborative systems. ACM Transactions on Computer-Human Interaction, 1994, 1(4): 295–330
22. Choudhary R, Dewan P. A general multi-user undo/redo model. In: Proceedings of European Conference on Computer-Supported Cooperative Work. 1995, 231–246
23. Sun C Z. Undo as concurrent inverse in group editors. ACM Transactions on Computer-Human Interaction, 2002, 9(4): 309–361
24. Ferrié J, Vidot N, Cart M. Concurrent undo operations in collaborative environments using operational transformation. In: Proceedings of OTM Confederated International Conference on the Move to Meaningful Internet Systems. 2004, 155–173
25. Weiss S, Urso P, Molli P. An undo framework for P2P collaborative editing. In: Proceedings of EAI International Conference on Collaborative Computing. 2009, 529–544
26. Weiss S, Urso P, Molli P. A flexible undo framework for collaborative editing. INRIA Research Report, 2008, RR-6516
27. Sun D, Sun C Z. Context-based operational transformation in distributed collaborative editing systems. IEEE Transactions on Parallel and Distributed Systems, 2009, 20(10): 1454–1470
28. Shao B, Li D, Gu N. An algorithm for selective undo of any operation in collaborative applications. In: Proceedings of ACM International Conference on Supporting Group Work. 2010, 131–140
29. Yoon Y S, Myers B A. Supporting selective undo in a code editor. In: Proceedings of International Conference on Software Engineering. 2015, 223–233
30. Cherif A, Imine A. Using CSP for coordinating undo-based collaborative applications. In: Proceedings of ACM Symposium on Applied Computing. 2016, 1928–1935
31. Ressel M, Nitsche-Ruhland D, Gunzenhaeuser R. Integrating, transformation-oriented approach to concurrency control and undo in group editors. In: Proceedings of ACM Conference on Computer Supported Cooperative Work. 1996, 288–297
32. Li R, Li D, Sun C Z. A time interval based consistency control algorithm for interactive groupware applications. In: Proceedings of IEEE International Conference on Parallel and Distributed Systems. 2004, 429–436
33. Xu Y, Sun C Z. Conditions and patterns for achieving convergence in OT-based co-editors. IEEE Transactions on Parallel and Distributed Systems, 2016, 27(3): 695–709
34. Lamport L. Time, clocks, and the ordering of events in a distributed system. Communications of the ACM, 1978, 21(7): 558–565
35. Li D, Li R. Preserving operation effects relation in group editors. In: Proceedings of ACM Conference on Computer Supported Cooperative Work. 2004, 457–466
36. Oster G, Molli P, Urso P, Imine A. Tombstone transformation functions for ensuring consistency in collaborative editing systems. In: Proceedings of EAI International Conference on Collaborative Computing. 2006, 1–10
37. Li R, Li D. A new operational transformation framework for real-time group editors. IEEE Transactions on Parallel and Distributed Systems, 2007, 18(3): 307–319
38. Imine A. Flexible concurrency control for real-time collaborative editors. In: Proceedings of IEEE International Conference on Distributed Computing Systems. 2008, 423–428
39. Li D, Li R. An admissibility-based operational transformation framework for collaborative editing systems. Computer Supported Cooperative Work, 2010, 19(1): 1–43
40. Gu N, Yang J M, Zhang Q W. Consistency maintenance based on the mark & retrace technique in groupware systems. In: Proceedings of ACM International Conference on Supporting Group Work. 2005, 264–273
41. Shao B, Li D, Gu N. A fast operational transformation algorithm for mobile and asynchronous collaboration. IEEE Transactions on Parallel and Distributed Systems, 2010, 21(12): 1707-1720
42. Sun C Z, Sun D, Ng A, Cai W W, Cho B. Real differences between OT and CRDT under a general transformation framework for consistency maintenance in co-editors. Proceedings of the ACM on Human-Computer Interaction, 2020, 4(GROUP): 1–26
43. Sun D, Sun C Z, Ng A, Cai W W. Real differences between OT and CRDT in correctness and complexity for consistency maintenance in co-editors. Proceedings of the ACM on Human-Computer Interaction, 2020, 4(CSCW1): 1-30
44. Sun D, Sun C Z, Ng A, Cai W W. Real differences between OT and CRDT in building co-editing systems and real world applications. 2020, arXiv preprint arXiv:1905.01517
45. Oster G, Urso P, Molli P, Imine A. Data consistency for P2P collaborative editing. In: Proceedings of ACM Conference on Computer Supported Cooperative Work. 2006, 259–268
46. Shapiro M, Preguiça N, Baquero C, Zawirski M. Conflict-free replicated data types. INRIA Research Report, 2011, RR-7687
47. Weiss S, Urso P, Molli P. Logoot-Undo: distributed collaborative editing system on P2P networks. IEEE Transactions on Parallel and Distributed Systems, 2010, 21(8): 1162–1174
48. Yu W H. Supporting string-wise operations and selective undo for peer-to-peer group editing. In: Proceedings of ACM International Conference on Supporting Group Work. 2014, 226–237
49. Li D, Li R. A performance study of group editing algorithms. In: Proceedings of IEEE International Conference on Parallel and Distributed Systems. 2006, 300–307
50. Lim Y, Ahn S. Architecture for mobile group communication in campus environment. Frontiers of Computer Science, 2013, 7(4): 505–513
51. Gao L P, Yu F Y, Chen Q K, Xiong N X. Consistency maintenance of do and undo/redo operations in real-time collaborative bitmap editing systems. Cluster Computing, 2016, 19(1): 255–267
52. Li H R, He F Z, Liang Y Q, Quan Q. A dividing-based many-objective evolutionary algorithm for large-scale feature selection. Soft Computing, 2020, 24(9): 6851–6870
53. Zhang S D, He F Z. DRCDN: learning deep residual convolutional dehazing networks. The Visual Computer, 2020, 36(9): 1797–1808
54. Cui Z Y, Liu Y, Zhao W. YUN: a fast ground-to-air cloud image recognition framework. In: Proceedings of IEEE International Conference on Computer Supported Cooperative Work in Design. 2019, 290–294.
55. Quan Q, He F Z, Li H R. A multi-phase blending method with incremental intensity for training detection networks. The Visual Computer, 2021, 37(2): 245–259
56. Yu H P, He F Z, Pan Y T. A scalable region-based level set method using adaptive bilateral filter for noisy image segmentation. Multimedia Tools and Applications, 2020, 79: 5743–5765
57. Gao M, Ling B, Yang L, Wen J H, Xiong Q Y, Li S. From similarity perspective: a robust collaborative filtering approach for service recommendations. Frontiers of Computer Science, 2019, 13(2): 231–246
58. Chen Y L, He F Z, Li H R, Zhang D J, Wu Y Q. A full migration BBO algorithm with enhanced population quality bounds for multimodal biomedical image registration. Applied Soft Computing, 2020, 93: 106335
59. Su K, Yang G P, Yang L, Su P, Yin Y L. Non-negative locality-constrained vocabulary tree for finger vein image retrieval. Frontiers of Computer Science, 2019, 13(2): 318–332
60. Yong J S, He F Z, Li H R, Zhou W Q. A novel bat algorithm based on cross boundary learning and uniform explosion strategy. Applied Mathematics—A Journal of Chinese Universities, 2019, 34(4): 482-504
61. Wu C X, Li L F, Peng C W, Wu Y, Xiong N X, Lee C. Design and analysis of an effective graphics collaborative editing system. Eurasip Journal on Image and Video Processing, 2019, 1: 50

62. Zhang J, He F Z, Chen Y L. A new haze removal approach for sky/river alike scenes based on external and internal clues. Multimedia Tools and Applications, 2020, 79: 2085–2107
63. Wang T, Zhang Q P, Liu Z, Liu W L, Wen D. On social computing research collaboration patterns: a social network perspective. Frontiers of Computer Science, 2012, 6(1): 122–130
64. Pan Y T, He F Z, Yu H P. Learning social representations with deep autoencoder for recommender system. World Wide Web, 2020, 23(4): 2259–2279
65. Shi X H, Lu H T. Community detection in scientific collaborative network with bayesian matrix learning. Frontiers of Computer Science, 2019, 13(1): 212–214
66. Pan Y T, He F Z, Yu H P. A correlative denoising autoencoder to model social influence for top-n recommender system. Frontiers of Computer Science, 2020, 14(3): 143301
67. Zhang G, Li Y M, Shi Y H. Distributed learning particle swarm optimizer for global optimization of multimodal problems. Frontiers of Computer Science, 2018, 12(1): 122–134
68. Luo J K, He F Z, Yong J S. An efficient and robust bat algorithm with fusion of opposition-based learning and whale optimization algorithm. Intelligent Data Analysis, 2020, 24(3): 581–606
69. Zhao W, Shen H H, Zhang F, Tan H Z. Adaptive power optimization for mobile traffic based on machine learning. In: Proceedings of IEEE International Conference on Computer Supported Cooperative Work in Design. 2019, 500–505
70. Luo J K, He F Z, Li H R, Z X T, Liang Y Q. A novel whale optimization algorithm with filtering disturbance and non-linear step. International Journal of Bio-Inspired Computation, 2020, 16: 137–148
71. Hou N, He F Z, Zhou Y, Chen Y L. An efficient GPU-based parallel tabu search algorithm for hardware/software co-design. Frontiers of Computer Science, 2020, 14(5): 145316
72. Zhang S Q, Qin Z, Yang Y H, Shen L, Wang Z Y. Transparent partial page migration between CPU and GPU. Frontiers of Computer Science, 2020, 14(3): 143101

Weiwei Cai is currently a PhD candidate at School of Computer Science of Wuhan University, China. His research interests include computer supported collaborative work and distributed computing systems.

Fazhi He is currently a professor at School of Computer Science of Wuhan University, China. His research interests include collaborative computing, computer graphics, computer vision, high-performance computing.



Xiao Lv is currently a lecturer at Department of Computer Engineering of Naval University of Engineering, China. Her research interests include computer supported collaborative work, collaborative design.

Yuan Cheng is currently a lecturer at School of Information Management of Wuhan University, China. Her research interests include computer supported collaborative work, scientific visualization, computation geometry.