

[文章编号] 1003-4684(2024)02-0010-07

基于冲突划分的位图多版本增量创建算法

熊才权, 陈伟杰, 吴歆韵

(湖北工业大学计算机学院, 湖北 武汉 430068)

[摘 要] 根据位图操作间的冲突关系对待执行操作在位图版本中的冲突像素区域进行冲突划分,将属于相同冲突的像素区域独立成一个冲突像素集。使用多版本方法解决冲突,即对于每一个冲突像素集,将位图版本增量复制后,分别在冲突像素集中所包含的像素区域执行冲突操作的操作效果得到增量创建的位图版本,以达到一致性维护的目的。最后对方法的正确性和有效性进行了证明。在自定义的数据集上进行实验,结果表明,所提出的 CDB-MVIC 算法产生的位图版本数量和冗余位图版本数量都比 BTMVIC 算法少。

[关键词] 位图; 协同图形编辑; 冲突划分; 多版本方法; 一致性维护

[中图分类号] TP18 **[文献标识码]** A

实时协同编辑系统利用计算机网络和通信技术支持不同地点的各个协作人员共同编辑共享的文档^[1]。协同编辑系统的主要特点是实时交互和分布式^[2]。实时协同图形编辑系统是协同编辑系统中的一种^[3]。在实时协同图形编辑系统中,多个用户针对同一个图形发出的多个操作可能会同时生成。由于网络环境的不确定性,各个操作在本地站点生成并发送到远程站点后,各站点执行这些操作的顺序可能会出现不一致,从而导致共享图形文档不一致的问题,用户的操作意图也不能完全得到保证^[4]。所以,一致性维护是一种保证协同图形编辑系统正确性的重要策略^[5]。现有的研究主要集中在基于对象的协同图形编辑上,而基于位图的协同图形编辑研究比较少,然而基于位图的协同编辑系统相较于基于对象的协同图形编辑系统在对图像的处理上有着显著的优势^[6],故此研究是有意义的。

在基于位图的协同图形编辑的一致性维护研究中,Wang 等^[7]引入有向图结构和点列表描述了位图操作之间的冲突关系,提出了一种 Any-undo 算法,在保证一致性的情况下实现了位图操作的撤销,但由于点列表会随着有向图结构的改变而变化,如果某些操作被错误地撤销了,整个有向图结构都会受到影响。Wang 等^[8]基于多版本思想,在对位图版本增量创建时对操作进行转换,使操作相容地存在于同一个位图版本中,并考虑了位图操作间颜色的重合。该方法虽然充分保留了用户的操作意图,

但少量操作冲突便可产生大量的位图版本,使得系统可用性差,位图版本的维护也较为困难。Myers 等^[9]开发了一个位图编辑系统(Aquamarine),可对位图操作进行选择性的撤销,但此系统并未考虑操作间的并发冲突。Gao 等^[10]提出了一种 BTMVIC 算法用于解决 Do 操作的一致性维护算法。该算法利用操作和历史操作集 HB 的关系对位图版本进行增量创建,但这种增创方式易产生大量冗余的位图版本。

本文对 BTMVIC 算法的位图版本增量创建方式进行了改进,提出一种称为 CDBMVIC 的位图多版本增创算法,该算法首先对待执行操作在位图版本中的执行像素区域进行冲突检测,若发生冲突,则存入集合中,然后依据操作间的冲突关系对集合中的像素进行冲突划分,对于每一个冲突像素集合,使用多版本方法首先对位图版本进行增量复制,然后分别在复制位图版本中执行冲突操作,即可得到此位图版本增量复制的结果,从而达到一致性维护的目的。

1 解决冲突的多版本方法

1.1 基本定义

定义 1 位图操作(Operation):位图操作是对文档中的像素设置颜色,表示为 Operation(SiteID, C_Location, Color),其中 SiteID 表示操作的站点标识符,C_Location 表示受操作影响的像素集,Col-

[收稿日期] 2022-08-19

[基金项目] 湖北省科技计划项目(2021BLB171);国家自然科学基金(61902116);湖北工业大学绿色工业科技引领计划项目(CPYF2017008)

[第一作者] 熊才权(1966-),男,湖北鄂州人,工学博士,湖北工业大学教授,研究方向人工智能,辩论模型,智能决策。

[通信作者] 陈伟杰(1997-),男,湖北黄冈人,湖北工业大学硕士研究生,研究方向为 CSCW,协同图形编辑,一致性维护。

or 表示操作执行的颜色。

定义 2 位图版本(BV)和位图版本集(BVS):一个位图文档对应一个位图版本,用 BV 来表示。位图版本集为多个位图版本的集合,表示为 BVS。

1.2 多版本方法

目前,多版本方法^[11]、操作转换思想^[12]和可交换的复制式数据类型^[13]是三类主要的一致性维护方法,这三种方法的实质都是通过控制操作的执行,解决操作间的并发冲突来达到一致性维护的目的。其中,多版本方法强调保留用户的操作意图,方法的核心思想是通过增加目标文档的版本数量来解决操作间的并发冲突。即当并发操作发生冲突时,从原始版本派生出多个复制版本,然后将这些冲突操作分别作用到不同的复制版本上^[14]。

在基于位图的协同图形编辑系统中,多版本方法也可用于解决操作之间的冲突。实现多用户协作时共享图形文档的一致性维护。具体来说: $O_a \otimes O_b$,两个操作冲突像素区域为 C ,将原位图版本复制成两个新位图版本 BV_1 和 BV_2 ,然后分别将 O_1 的操作效果和 O_2 的操作效果分别应用在新位图版本 BV_1 和 BV_2 的 C 区域,使冲突操作在冲突区域的操作效果呈现在不同的位图版本中。例如:如图 1 所示,假设存在操作 $O_1 = \text{Operation}(\text{Site1}, \{(1, 1), (2, 1), \text{red}\})$ 和操作 $O_2 = \text{Operation}(\text{Site2}, \{(0, 1), (1, 1)\}, \text{green})$, $O_1 \otimes O_2$ 且 $O_1 \cap O_2 = \{(1, 1)\} = C$,那么根据多版本方法则会生成两个位图版本 BV_1 和 BV_2 来解决 O_1 和 O_2 的操作冲突。

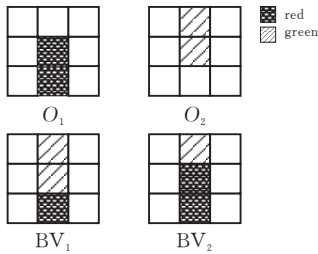


图 1 操作冲突处理策略

2 BTMVIC 算法的不足

BTMVIC 算法是现有的基于多版本方法解决基于位图的协同图形编辑中的一致性维护算法。设一组位图操作 $GO = \{O_1, O_2, \dots, O_n\}$,以 O_1, O_2, \dots, O_n 的顺序依次执行, BVS_n 是执行 n 个操作后生成的位图版本集,HB 中存放已经执行过的操作。BTMVIC 算法主要描述的是根据操作 O_i 与 HB 中操作的因果并发关系对位图版本集 BVS_{i-1} 进行增量创建生成 BVS_i 的过程。位图操作间的因果并发关系可参见文献[10]。

BTMVIC 算法使用多版本方法解决操作冲突的方法为:遍历 HB,每检测到一个与 O_i 冲突的操作 O_j ,就将所有位图版本复制两份,分别在两个复制的位图版本 $O_i \cap O_j$ 像素区域执行 O_i 和 O_j 的操作效果。这样增量创建的位图版本存在如下两个问题。

1)算法每检测到一个冲突,就将所有位图版本进行增创来解决冲突,故算法增量创建的位图版本数量呈指数级增长。如果冲突数为 n ,则 BTMVIC 算法会以 2^n 的数量级对位图版本进行增量创建。且如果有些历史操作的操作效果已经被其他操作的操作效果覆盖了,但该操作依然与待执行操作发生冲突,那么此次对位图版本的增量创建是多余的。

2)算法适用于解决冲突间的冲突像素区域不存在交集的情况。但当冲突像素区域间存在交集时,则容易产生位图版本的冗余。假设待执行操作检测到最后一个与其冲突的操作,其冲突像素区域包含其它所有冲突的冲突像素区域,则对当前位图版本进行增量创建后,仅有两个不同的位图版本,并且这两个位图版本存在较大的冗余。

本文所提出的 CDBMVIC 算法改变了 BTMVIC 算法的增量创建方式,通过对待执行操作在位图版本的执行像素区域进行冲突划分后能够明确待执行操作在当前位图版本中的冲突情况,能够有效减少对位图版本的增量创建次数。且冲突像素划分后能够将每个冲突的冲突像素集合独立起来,不会在位图版本增量创建后产生位图版本的冗余问题。

3 CDBMVIC 算法

3.1 冲突划分

当对某一位图版本执行操作 O_i 时,首先需要冲突检测函数 checkConflict ^[8] 检测待执行操作与历史操作中的哪些操作会发生冲突,若检测到操作 O_i 与历史操作中某一操作 O_j 发生冲突,则在待执行操作在该位图版本中的执行像素区域检索出属于该冲突的冲突像素,从而得到冲突划分的结果。采用算法 conflictDivision 对冲突像素进行划分后得到集合 PixelsDivision 描述了此过程。 PixelsDivision 中元素 ConflictEle 的个数为冲突数,每一个 ConflictEle 是一个三元组 $(O_j, O_i, \text{Pixels})$,表示待执行操作 O_i 在像素集合 Pixels 与历史操作 O_j 在位图版本中发生的冲突, conflictDivision 算法描述如下。

算法 1. $\text{conflictDivision}(O_i, BV, HB) : \text{PixelsDivision}$

输入:待执行的操作 O_i ;位图版本 BV ;历史操作集合 HB

输出: PixelsDivision

```

1) For  $O_j$  in HB
2) ConflictPixels = {}
3) If checkConflict( $O_i, O_j$ ) == true then
4) ConflictPixels =  $O_i \cap O_j$  // Conflict-
   Pixels 为操作  $O_i$  和  $O_j$  的冲突像素集合
5) End if
6) Pixels = {}
7) For pixel in C // C 为待执行操作  $O_i$  在
   位图版本 BV 中的执行像素区域
8) If pixel  $\in$  ConflictPixels then
9) Pixels.add(pixel)
10) End if
11) End for
12) ConflictEle = ( $O_j, O_i, \text{Pixels}$ )
13) PixelsDivision.add(ConflictEle)
14) End for
15) return PixelsDivision

```

3.2 基于多版本方法的冲突解决策略

将待执行操作在位图版本中的冲突划分完成后,使用多版本方法解决冲突。解决每一个冲突时,均需要将位图版本复制两份,分别在复制版本的冲突像素区域执行冲突操作的操作效果。即可解决待执行操作在位图版本中的所有冲突。使用 SetColor 函数^[10]实现在位图中的某像素区域执行操作的操作效果。算法 2 描述了此过程。

算法 2. ConflictResolve($O_i, BV, \text{PixelsDivision}$): BVS

输入:待执行的操作 O_i ;位图版本 BV;冲突像素集合 PixelsDivision

输出:位图版本集 BVS

```

1) BVStemp.add(BV)
2) For ConflictEle in PixelsDivision
3) BVS = {}
4) For BV in BVStemp
5) Copy BV as BVnew1, BVnew2
6) Execute {SetColor(Pixels,  $O_i$ );} on
   BVnew1
7) Execute {SetColor(Pixels,  $O_j$ );} on
   BVnew2
8) BVS.add(BVnew1)
9) BVS.add(BVnew2)
10) End for
11) BVStemp = BVS
12) End for
13) BVS = BVStemp

```

14) return BVS

3.3 CDBMVIC 算法流程

设 BVS_{i-1} 为一组操作 $GO = \{O_1, O_2, \dots, O_{i-1}\}$ 执行后产生的位图版本集, O_i 是待执行的位图操作。下面具体描述此算法:

算法 3. CDBMVIC(O_i, BVS_{i-1}): BVS_i

输入:待执行的操作 O_i ; $GO = \{O_1, O_2, \dots, O_{i-1}\}$ 中的所有操作执行后生成的位图版本集 BVS_{i-1}

输出: BVS_i

```

1) If BVSi-1 == null then
2) BVSi = SetColor( $C_1, O_i$ ); //  $C_1$  为待执行
   操作  $O_i$  的执行像素区域
3) Else
4) For BVi in BVSi-1
5) PixelsDivision = conflictDivision( $O_i, BV_i, HB$ )
6) BVStemp = ConflictResolve( $O_i, BV_i, \text{PixelsDivision}$ )
7)  $C = C_1 - \text{PixelsDivision}$ ;
8) If  $C \neq \Phi$  then
9) Execute {SetColor( $C, O_i$ );} on every
   BV in BVStemp
10) End if
11) BVSi.add(BVStemp)
12) End for
13) End if
14) return BVSi

```

CDBMVIC 算法描述了将待执行操作 O_i 应用于位图版本集 BVS_{i-1} 增量创建 BVS_i 的过程。当执行第一个操作时,此时位图版本集为空,函数 SetColor 在位图中执行第一个操作并生成第一个位图版本(1~2 行)。否则,对 BVS_{i-1} 中的每一个位图版本 BV_i ,通过 conflictDivision 算法确定冲突的个数并得到 PixelsDivision 集合来表示冲突情况,然后执行 ConflictResolve 算法利用多版本方法对位图版本进行增量创建来解决操作冲突。最后在所有增量创建的位图版本中操作 O_i 与原位图版本不冲突的像素区域执行 O_i 的操作效果存入 BSV_i 中。(4~14 行)。

此算法的时间复杂度主要取决于位图版本的数量、历史操作集 HB 中操作的个数和待执行操作所影响的像素个数。若位图版本为 m , HB 中操作的个数为 n ,待执行操作影响的像素个数为 r ,则算法的时间复杂度为 $O(m \times n \times r)$ 。

3.4 CDBMVIC 算法正确性证明

定理 1 给定一组位图操作 $GO=\{O_1,O_2,\cdots,O_n\}$,在保证操作因果一致性的情况下,以 O_1,O_2,\cdots,O_n 的顺序依次执行,由 CDBMVIC 算法得到的位图版本集 BVS_n 是 GO 唯一的 BVS 。

证明 采用数学归纳法证明。当 $n=1$ 时,操作组 $GO_1=\{O_1\}$,生成一个位图版本,没有冲突,此时 BVS_1 是 GO_1 唯一的 BVS 。

假设 当 $n=m$ 时, BVS_m 是 $GO_m=\{O_1,O_2,\cdots,O_m\}$ 唯一的 BVS 。

当 $n=m+1$ 时, $GO_{m+1}=\{O_1,O_2,\cdots,O_m,O_{m+1}\}$,根据 CDBMVIC 算法,对 BVS_m 中的每一个 BV 执行操作 O_{m+1} 。又由于 BVS_m 是 GO_m 唯一的 BVS ,故执行完成操作 O_{m+1} 后所得到的 BVS_{m+1} 是 GO_{m+1} 唯一的 BVS 。

综上所述,当 $n=m+1$ 时,定理成立。

定理 2 给定一组位图操作 $GO=\{O_1,O_2,\cdots,O_n\}$,在保证操作因果一致性的情况下,无论操作以什么次序执行,由 CDBMVIC 算法得到的 BVS 都是一致的。

证明 在一致性准则中,对操作执行顺序唯一的约束是因果操作,为了在各个站点保证操作间的因果一致性,通常采用时间戳的方法实现因果操作间的因果保留^[10]。在因果一致性的情况下,由定理 1 可知存在唯一的 BVS 与 GO 对应,且当操作以 O_1,O_2,\cdots,O_n 的顺序依次执行,由 CDBMVIC 算法得到的位图版本集 BVS_n 是 GO 的 BVS 。故由定理 1 可直接得到此定理。

定理 3 CDBMVIC 算法能够保证用户的操作意图。

证明 CDBMVIC 算法基于多版本方法实现用户操作意图的保留。当两个操作冲突时,通过版本复制的方式同时保留两个冲突操作的操作意图。当操作与位图版本存在多个操作冲突时,通过冲突划分使得每个操作冲突独立起来,则进行位图版本增量复制的时候能够保证每一个冲突操作都至少存在一个位图版本,能够完全保留该操作的执行效果,即保证了用户的操作意图。

基于以上三个定理,在基于位图的协同图形编辑中,如在协同环境中出现操作冲突时,CDBMVIC 算法能够解决操作冲突并实现各个站点的一致性维护,同时能够保证用户的操作意图。

3.5 案例分析

图 2 展示了位图版本的初始状态和操作 O_1 、 O_2 、 O_3 、 O_4 的执行效果。

如图 3 所示,操作 O_1 和 O_2 在站点 1 生成,操作

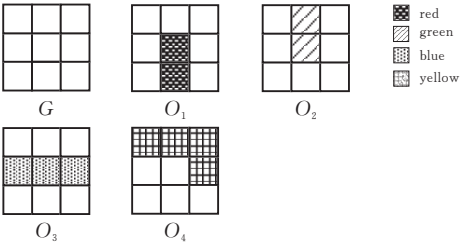


图 2 用户操作

O_3 和 O_4 在站点 2 生成,即 $O_1 = \text{Operation}(\text{Site1}, \{(1,1), (2,1)\}, \text{red})$, $O_2 = \text{Operation}(\text{Site1}, \{(0,1), (1,1)\}, \text{green})$, $O_3 = \text{Operation}(\text{Site2}, \{(1,0), (1,1), (1,2)\}, \text{blue})$, $O_4 = \text{Operation}(\text{Site2}, \{(0,0), (0,1), (0,2), (1,2)\}, \text{yellow})$ 。位图操作间的关系为: $(O_1 \rightarrow O_2) \parallel O_3, O_1 \rightarrow (O_2 \parallel O_4), O_3 \rightarrow O_4$, 其中 $O_1 \otimes O_3, O_2 \otimes O_3, O_2 \otimes O_4$ 。分别在站点 1 和站点 2 详细分析操作的执行过程,执行结果如图 4 所示。

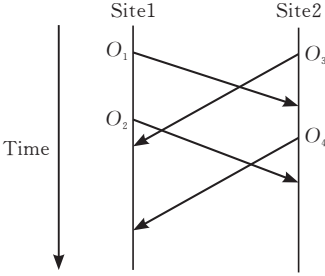


图 3 案例分析

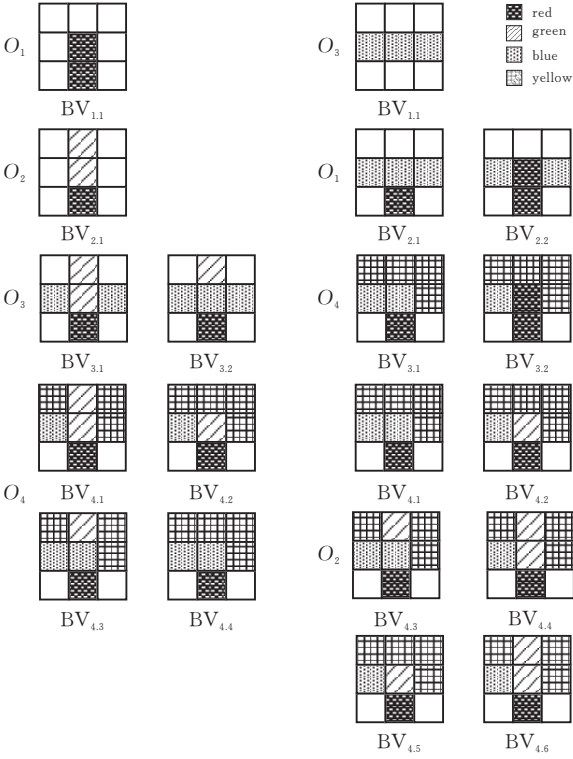


图 4 不同操作执行顺序的操作效果

站点 1:操作执行顺序为 O_1,O_2,O_3,O_4 :

1)操作 O_1 在站点 1 上生成并立即执行,生成

位图版本 $BV_{1,1}$

2) 当操作 O_2 在站点 1 生成时, 位图版本 $BV_{1,1}$ 中只存在 O_1 的操作效果, 由于 $O_1 \rightarrow O_2$, 故直接将 O_2 的操作效果覆盖在位图版本 $BV_{1,1}$ 之上, 生成位图版本 $BV_{2,1}$ 。

3) 当站点 1 接收到来自站点 2 的操作 O_3 时, 此时操作 O_3 只在位图版本 $BV_{2,1}$ 上的像素 $(1,1)$ 发生冲突, 且此冲突为操作 O_2 和 O_3 的冲突, 故 $\text{PixelsDivision} = \{(O_2, O_3, \{(1,1)\})\}$, 将位图版本 $BV_{2,1}$ 复制两份, 分别在像素集 $\{(1,1)\}$ 执行操作 O_2 和 O_3 的操作效果。由于 O_3 与位图版本 $BV_{2,1}$ 中的其他像素不发生冲突, 故在其它像素直接执行 O_3 的操作效果即可, 生成位图版本 $BV_{3,1}, BV_{3,2}$ 。

4) 当操作 O_4 到达站点 1 时, 对于位图版本 $BV_{3,1}$, 此时操作 O_4 只在像素 $(0,1)$ 发生冲突, 且此冲突为操作 O_2 和 O_4 的冲突, 故 $\text{PixelsDivision} = \{(O_2, O_4, \{(0,1)\})\}$, 将位图版本 $BV_{3,1}$ 复制两份, 分别在像素集 $\{(0,1)\}$ 执行操作 O_2 和 O_4 的操作效果。同理, 对于位图版本 $BV_{3,2}$ 也执行相同的操作。最后, 由于 O_4 在位图版本 $BV_{3,1}$ 和位图版本 $BV_{3,2}$ 中的其它像素不发生冲突, 故在所有位图版本中的其它像素区域直接执行 O_4 的操作效果即可, 生成位图版本 $BV_{4,1} \sim BV_{4,4}$ 。

站点 2: 操作执行顺序为 O_3, O_1, O_4, O_2 :

1) 操作 O_3 在站点 1 上生成并立即执行, 生成位图版本 $BV_{1,1}$

2) 当接收到来在站点 1 的操作 O_1 时, 此时位图版本 $BV_{1,1}$ 上只有 O_3 的操作效果, 对于位图版本 $BV_{1,1}$, 操作 O_1 只在像素 $(1,1)$ 发生冲突, 且此冲突为操作 O_1 和 O_3 的冲突, 故 $\text{PixelsDivision} = \{(O_3, O_1, \{(1,1)\})\}$, 将对位图版本 $BV_{1,1}$ 复制两份, 分别在像素集 $\{(1,1)\}$ 执行操作 O_3 和 O_1 的操作效果。然后, 由于 O_1 在位图版本 $BV_{1,1}$ 中的其它像素不发生冲突, 故在其它像素区域直接执行 O_3 的操作效果即可, 生成位图版本 $BV_{2,1}, BV_{2,2}$ 。

3) 当站点生成操作 O_4 并执行时, 操作 O_4 在所有位图版本中的像素区域均不发生冲突, 故对所有位图版本直接执行 O_4 的操作效果, 生成位图版本 $BV_{3,1}, BV_{3,2}$ 。

4) 当操作 O_2 到达站点 2 后, 对位图版本 $BV_{3,1}$ 执行操作 O_2 , 此时 $\text{PixelsDivision} = \{(O_4, O_2, \{(0,1)\}), (O_3, O_2, \{(1,1)\})\}$, 首先对于冲突 $(O_4, O_2, \{(0,1)\})$, 将对位图版本 $BV_{3,1}$ 复制两份, 分别在像素集 $\{(0,1)\}$ 执行操作 O_4 和 O_2 的操作效果。然后对于冲突 $(O_3, O_2, \{(1,1)\})$, 将当前所有的位图版本复制两份, 分别在像素集 $\{(1,1)\}$ 执行操作 O_3 和

O_2 的操作效果, 生成位图版本 $BV_{4,1} \sim BV_{4,4}$ 。对于位图版本 $BV_{3,2}$, 操作 O_2 只在像素 $(0,1)$ 发生冲突, 且此冲突为操作 O_2 和 O_4 的冲突, 故 $\text{PixelsDivision} = \{(O_4, O_2, \{(0,1)\})\}$, 故对位图版本 $BV_{3,2}$ 复制两份, 分别在像素集 $\{(0,1)\}$ 执行操作 O_4 和 O_2 的操作效果。然后对未发生冲突的像素区域直接执行操作 O_2 的操作效果, 生成位图版本 $BV_{4,5} \sim BV_{4,6}$ 。此时, 位图版本 $BV_{4,2}$ 与 $BV_{4,5}$ 相同, 位图版本 $BV_{4,4}$ 与 $BV_{4,6}$ 相同, 去冗后即可得到最终的位图版本 $BV_{4,1} \sim BV_{4,4}$, 且与站点 1 所增量创建的位图版本是一致的。

4 模拟实验

在 Windows 10 操作系统上, 使用 IntelliJ IDEA 作为实验平台。用 Java 语言实现了 CDBMVIC 算法和 BTMVIC^[10] 算法并进行对比。如图 5 所示, 假设四个站点分别生成四个操作 $O_1 = \text{Operation}(\text{Site1}, \{(1,1), (2,1)\}, \text{red})$ 、 $O_2 = \text{Operation}(\text{Site2}, \{(1,1), (1,2)\}, \text{green})$ 、 $O_3 = \text{Operation}(\text{Site3}, \{(0,1), (1,1)\}, \text{blue})$ 、 $O_4 = \text{Operation}(\text{Site4}, \{(1,0), (1,1)\}, \text{yellow})$, 操作执行顺序为 O_1, O_2, O_3, O_4 。每两个操作之间均存在冲突关系, 且冲突像素均为 $\{(1,1)\}$ 。设计冲突用例分别进行 BTMVIC 算法和 CDBMVIC 算法的模拟试验, 并通过比较两个算法位图版本增创的数量 c 和冗余位图版本的数量 r 来比较算法的优劣。

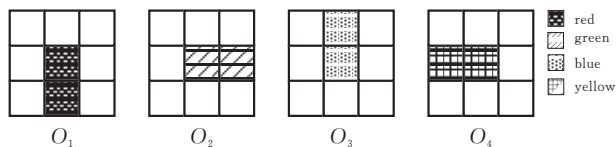


图 5 用户操作

冲突用例 1: $(O_1 \otimes O_2) \rightarrow O_3; (O_1 \otimes O_2) \rightarrow O_4; O_3 \rightarrow O_4$

冲突用例 2: $(O_1 \otimes O_2) \rightarrow O_4; (O_1 \otimes O_3) \rightarrow O_4; O_2 \rightarrow O_3$

冲突用例 3: $(O_1 \otimes O_2 \otimes O_3) \rightarrow O_4$

冲突用例 4: $O_1 \otimes (O_2 \rightarrow O_4); O_1 \otimes (O_3 \rightarrow O_4); O_2 \otimes O_3$

表 1 展示了各个冲突用例的实验结果。对于冲突用例 1, 此冲突用例中只存在一个操作冲突, CDBMVIC 算法增创的位图版本数量和冗余位图版本数量与 BTMVIC 算法相同。冲突用例 2 中存在两个操作冲突, BTMVIC 算法增创的位图版本数量为 4, 其中有 3 个位图版本是冗余的, 而 CDBMVIC 算法的位图版本增创数量为 3, 其中有 2 个位图版本是冗余的。冲突用例 3 中存在三个操作冲突, BT-

MVIC 算法增创的位图版本数量为 8,冗余的位图版本数量为 7,而 CDBMVIC 算法的位图版本增创数量为 4,冗余的位图版本数量为 3。最后一个冲突用例的冲突数为 4,BTMVIC 算法增创的版本数量为 16,其中,冗余的位图版本数量为 14,而 CDBMVIC 算法增创的位图版本数量为 5,其中,冗余的位图版本数量为 3。通过比较 CDBMVIC 算法和 BTMVIC 的位图版本增创数量和冗余的位图版本数量,结果表明,随着冲突数的增加,CDBMVIC 算法能够有效减少 BTMVIC 算法中位图版本的冗余创建。

表 1 BTMVIC 算法和 CDBMVIC 算法版本数量比较

冲突用例	冲突数量	BTMVIC		CDBMVIC	
		(c)	(r)	(c)	(r)
1	1	2	1	2	1
2	2	4	3	3	2
3	3	8	7	4	3
4	4	16	14	5	3

5 总结与展望

本文提出一种基于冲突划分的位图多版本增量创建算法 CDBMVIC,在保证用户操作意图的情况下,使用多版本方法,通过解决待执行操作在位图版本中像素区域的操作冲突对位图版本进行增量创建,实现了基于位图的协同图形编辑的一致性维护。给出相关定理和实例验证了算法的有效性和正确性。与现有的 BTMVIC 算法比较,本文所提出的算法能够有效地减少位图版本增量创建的冗余数量。后续研究主要集中在以下几个方面:首先,本文的算法仅适用于用户的 do 操作,以后可探索 undo 和 redo 操作;其次,考虑位图中操作的语义,探寻解决语义冲突的一致性维护算法;最后,尝试以算法为基础开发出基于位图的协同图形编辑系统。

[参 考 文 献]

[1] MONA A, ASMA C, ABDESSAMAD I. EdgeDoc: an edge-based distributed collaborative editing system [J]. *Pervasive and Mobile Computing*,2021,70(05):1-20.

[2] BATH ULRIKE, SHEKHAR SUMIT, DÖLLNER JÜRGEN,et al.COLiER: collaborative editing of raster images [C].//2021 International Conference on Cyberworlds (CW),2021:33-40.

[3] WU CHUN XUE,LI LANG FENG,PENG CHANG WEI,et al.Design and analysis of an effective graphics collaborative editing system [J]. *EURASIP Journal on Image and Video Processing*,2019,2019(01):1-21.

[4] 何发智,吕晓,蔡维纬,等.支持操作意图一致性的实时协同编辑算法综述[J].*计算机学报*,2018,41(04):840-867.

[5] GAO LI PING,GAO DONG FANG,XIONG NAI XUE,et al. CoWebDraw: a real-time collaborative graphical editing system supporting multi-clients based on HTML5 [J]. *Multimedia Tools and Applications*,2018,77(04):5067-5082.

[6] GAO LI PING,YU FANG YU,FU QIONG QIONG,et al.Undo/Redo operations in bitmap-based collaborative graphic editing systems [C].//International Conference on Human Centered Computing,2015:501-511.

[7] WANG XUE YI,BU JIA JUN,CHEN CHUN.Achieving undo in bitmap-based collaborative graphics editing systems [C].// Proceedings of the 2002 ACM Conference on Computer supported cooperative work,2002:68-76.

[8] WANG SHAN SHAN,WU CHUN XUE,GAO LI PING,et al. Research on consistency maintenance of the real-time image editing system based on bitmap [C].//Proceedings of the 2014 IEEE 18th International Conference on Computer Supported Cooperative Work in Design (CSCWD),2014:689-694.

[9] MYERS BRAD A,LAI ASHLEY,LE TAM MINH,et al. Selective undo support for painting applications[C].//Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems,2015:4227-4236.

[10] GAO LI PING,YU FANG YU,CHEN QING KUI,et al.Consistency maintenance of do and undo/redo operations in real-time collaborative bitmap editing systems [J]. *Cluster Computing*,2016,19(01):255-267.

[11] SUN CHENG ZHENG,CHEN DAVID.Consistency maintenance in real-time collaborative graphics editing systems [J]. *ACM Transactions on Computer-Human Interaction (TOCHI)*,2002,9(01):1-41.

[12] 许坚,姜晓峰,张坤.基于图形对象的一致性维护问题的研究 [J]. *计算机应用与软件*,2012,29(02):5.

[13] 吕晓,苑佳存,贾可荣,等.移动协同编辑中基于 CRDT 的序列转换算法[J].*华中科技大学学报(自然科学版)*,2022,50(02):130-135.

[14] 杨君,窦万峰.一种新的多版本增创算法[J].*计算机学报*,2008,31(04):702-710.

Bitmap Multiple Versions Incremental Creation Algorithm Based on Conflict Division

XIONG Caiquan¹, CHEN Weijie², WU Xinyun¹

(School of Computer Science, Hubei Univ. of Tech., Wuhan 430068, China)

Abstract: This paper proposes a Conflict Division Bitmap Multiple Versions Incremental Creation (CDBM-VIC) algorithm based on conflict division. According to the conflict relationship between bitmap operations, the conflicting pixel area in the bitmap version of the operation to be executed is conflicted and divided, separating the pixel regions belonging to the same conflict into a conflict pixel set. The multi-version method is used to resolve conflicts, that is, for each conflicting pixel set, to incrementally copy the bitmap version and perform conflicting operations on the pixel areas contained in the conflicting pixel set. Finally, the correctness and effectiveness of the method are proved. Experiments are carried out on a custom dataset, and the results show that the number of bitmap versions and redundant bitmap versions generated by the proposed CDBMVIC algorithm is less than that of the BTMVIC algorithm.

Keywords: bitmap; collaborative graphic editing; conflict division; multiple versions method; consistency maintenance

[责任编辑：张岩芳]

(上接第 4 页)

Research on Norm Optimal Iterative Learning Control Based on Data Driven

XU Wan, XIAO Di, CHEN Tingwei

(School of Mechanical Engineering, Hubei Univ. of Tech., Wuhan 430068, China)

Abstract: The traditional norm optimal iterative learning control (NOILC) can effectively improve the tracking accuracy of the servo system under the premise of determining the system model. However, in the actual control process, the system model parameters are often changed, resulting in a decline in the performance of the controller. Therefore, a data-driven norm-optimal iterative learning control method is proposed. Firstly, based on the input and output of the system, the cost function of the system estimation model is established. Then, the cost function is processed by partial differential, and a data-driven non-parametric model identification method is obtained. Finally, the model identification method is combined with NOILC. The experimental results show that for the time-varying system, the tracking error of this control method is 57.1 % of NOILC, and it converges five times ahead of NOILC. Therefore, the proposed method can effectively improve the tracking performance of time-varying systems.

Keywords: iterative learning; data drive; norm optimal; motion control

[责任编辑：张 众]