Nancy-Université
Université Henri Poincaré

AVERTISSEMENT

Ce document est le fruit d'un long travail approuvé par le jury de soutenance et mis à disposition de l'ensemble de la communauté universitaire élargie.

Il est soumis à la propriété intellectuelle de l'auteur. Ceci implique une obligation de citation et de référencement lors de l'utilisation de ce document.

D'autre part, toute contrefaçon, plagiat, reproduction illicite encourt une poursuite pénale.

➢ Contact SCD Nancy 1 : theses.sciences@scd.uhp-nancy.fr

LIENS

Code de la Propriété Intellectuelle. articles L 122. 4
Code de la Propriété Intellectuelle. articles L 335.2- L 335.10
http://www.cfcopies.com/V2/leg/leg_droi.php
http://www.culture.gouv.fr/culture/infos-pratiques/droits/protection.htm

Nancy-Université
Université Henri Poincaré

Département de formation doctorale en informatique **École doctorale IAEM Lorraine**
UFR STMIA

Memory optimization strategies for linear mappings and indexation-based shared documents

THÈSE

présentée et soutenue publiquement le November 14, 2011

pour l'obtention du

Doctorat de l'université Henri Poincaré – Nancy 1

(spécialité informatique)

par

M. Mumtaz AHMAD

Composition du jury

Rapporteurs :	Denis LUGIEZ	Professeur, Université de Provence, Marseille, FR
	Miki (Nicolas) HERMANN	CR (CNRS), École Polytechnique, Paris, FR
Examinateurs :	Eugeniusz Adam CICHON	Professeur, Université Henri Poincaré, Nancy, FR
	Siva ANANTHARAMAN	Professeur, Université d'Orléans, Orleans, FR
	Abdessamad IMINE	Maître de conférences, Université Nancy 2, FR
	Lhouari NOURINE	Professeur, Université Blaise Pascal, Aubière, FR
	Nacer BOUDJLIDA	Professeur, Université Henri Poincaré, Nancy, FR
	Serge BURCKEL	Maître de conférences, Université de la Réunion, FR

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Loria
Laboratoire lorrain de recherche en informatique et ses applications

Inria
informatics mathematics

I dedicate this thesis
to my
benefactors

Contents

General Introduction **xiii**

Chapter 1
Introduction **1**

1.1 Sequential Break-down of Operations 1
1.1.1 Optimization Context 3
1.2 Decentralized Collaborative Editing 4
1.2.1 Collaborative Editing Context 6
1.2.2 Approach by Weiss et al. 7
1.2.3 Approach by Preguiça et al. 12
1.3 Contributions 15
1.4 Overview and structure 16

Part I Sequential Break-Down of Operations **19**

Chapter 2
Compiler, Processor Optimization (A quick overview)

2.1 Compilers . . . 22
2.1.1 Compiler Optimization . . . 23
2.2 Processors . . . 24
2.2.1 Processor Optimization . . . 26
2.3 Applications . . . 26
2.4 Motivations and Objectives . . . 27

Chapter 3
***In Situ* Design of Computation for Linear Mappings over Fields**

3.1 Sequential Computation . . . 30
3.1.1 *In Situ* Design of Computation (IDC) . . . 31
3.2 Existence of IDC for Linear Mappings over Fields . . . 33
3.2.1 Case-I . . . 35
3.2.2 Case-II . . . 38
3.2.3 Computing Inverse Mapping . . . 39
3.2.4 Case-III . . . 39
3.2.5 Possibility of Computing Inverse Mappings . . . 41
3.3 An approach using Bézout's Identity . . . 41
3.3.1 Case-I . . . 43
3.3.2 Case-II . . . 43
3.3.3 Computing Inverse Mappings: . . . 45

Chapter 4
***In Situ* Design of Computation for Linear Mappings over Rings**

4.1 Existence of IDC for Linear Mappings over Rings . . . 48
4.1.1 Assignment Matrices . . . 48
4.2 Explanation and Construction . . . 50
4.2.1 Case-I . . . 50
4.2.2 Case-II . . . 52
4.3 Computing Inverse Mappings . . . 54

4.3.1 Possibility of computing Inverse mappings 55

Chapter 5
***In Situ* Design of Computation for Linear Mappings over Integers**

5.1 Existence of IDC for Linear Mappings over Integers 58
5.1.1 Modifications: 60
5.1.2 Explanation and construction of assignments: 60
5.2 Investigating bounds for the number of assignments: 68
5.2.1 An approach based on Fibonacci numbers 71
5.2.2 Identity (coefficient linkage): 74

Chapter 6
***In Situ* Design of Computation for Boolean Mappings**

6.1 Boolean Mappings 78
6.2 Computing Bijective Boolean Mappings 79
6.2.1 Explanation 81
6.2.2 Linearity Property 83
6.3 Computing General Boolean Mappings 84
6.3.1 Simple Mappings 85
6.3.2 Decomposing general boolean mappings 86
6.4 IDC for Polynomials over GF(2) 91
6.4.1 A First Tool 92
6.4.2 Explanation and Construction 95

Part II Decentralized Collaborative Editing 101

Chapter 7
Collaborative Editing System

7.1 Introduction 103

7.1.1 A Collaborative Editing Scenario 104
7.1.2 Pessimistic Concurrency Control 105
7.1.3 Optimistic Concurrency Control 106
7.1.4 Group Communication 106
7.1.5 Centralized collaborative editing systems 106
7.1.6 Decentralized collaborative editing systems 108
7.2 DCE Model 111
7.2.1 Framework Description 113
7.2.2 Editing a Document 114
7.2.3 Modifying a Document 115
7.2.4 Mapping between elements and Identifiers 115

Chapter 8
Generating Identifiers

8.1 Creating Unique Identifiers 118
8.1.1 Precision Control Technique 118
8.1.2 Generating user/site identifiers (*USIDs*) 120
8.1.3 The Procedure 121
8.1.4 Explanation 124
8.1.5 Uniqueness of line/character identifiers (*LCIDs*) . . . 125
8.1.6 Verifying uniqueness 127
8.2 Computing cardinality 127
8.2.1 Local cardinality 128
8.2.2 Global cardinality 128

Chapter 9
Properties and Analysis

9.1 Properties 132
9.1.1 Bijection property 132
9.1.2 Closure properties 133
9.2 Analysis 137
9.2.1 Comparison under Rounding Conditions 137
9.2.2 Comparison without Rounding Conditions 138
9.2.3 Comparison under floating-point arithmetics 138
9.2.4 Effect of different rounding conditions 139

9.2.5 Assuring order preservation 147

Chapter 10
Exchange of points and variation in global cardinality

10.1 Delayed exchange of points 150
10.1.1 Random Participation 150
10.1.2 Ordered Participation 154
10.2 Real time exchange of points 158
10.2.1 Computing Instant Global Cardinality 159
10.2.2 Comparison 160

Part III Evaluation **161**

Chapter 11
Further work and conclusion

11.1 Further work 163
11.1.1 Dispersion 163
11.1.2 One of the possible situations 165
11.1.3 Case-1 165
11.1.4 Case-2 165
11.1.5 Modifications and extensions of *IDC* 166
11.2 Conclusion 167

Appendixes

Appendix A
Experimentation Frame Work

A.1 Exchange of points (an experiment with 900 Users) 169
A.1.1 Delayed exchange of points with random participation 169
A.1.2 Delayed exchange of points with ordered participation 180

Appendix B
Implementation Framework

B.1 Algorithm 191

Glossary **195**

Index **197**

Bibliography **199**

List of Figures

1.1 Identifiers exchange scenario 9
1.2 Redundancy of Identifiers 11
1.3 Generating POSID 14

2.1 Compiler Interaction 22
2.2 Microprocessor 25
2.3 Intel®Core™ i7 processor 27

3.1 Sequential break-down of operations 31
3.2 Interpreting linear assignment 32
3.3 Reference value of linear assignment 33

6.1 Interpretation of "A First Tool" 95

7.1 A collaborative editing scenario 104
7.2 An example of collaborative editing 105
7.3 Wrong Integration 109
7.4 Correct Integration 110
7.5 Model 111
7.6 Inserting single character 112

8.1 Precision control principle 119
8.2 Position of an identifier in the interval 122
8.3 Insertion of points by single user 124

9.1 A scenario of three users 148

10.1 Users participation 153
10.2 Number of users vs global cardinality 153
10.3 Users participation 157
10.4 Number of users vs global cardinality 157

11.1 General diagram for dispersion problem 164

A.1 A view of users participation 179

A.2 Updates in global cardinality 180
A.3 Updates in global cardinality 190
A.4 A users participation curve 190

List of Algorithms

1 Function (How to generate new Line Identifier) 8
2 Function (How to Construct new ID) 8
3 Function (Generating POSID) 13

4 Boolean Compatible Decomposition 90

5 Function (Round a value) 119
6 How to generate *USIDs* 120
7 Function *middle*(How to compute new identifier) 122
8 How to generate a set of *LCIDs* 123
9 127

10 How to exchange points in Batch Mode 151
11 Batch Mode Ordered Participation 155

General Introduction

« Mathematical discoveries, small or great are never born of spontaneous generation. They always presuppose a soil seeded with preliminary knowledge and well prepared by labour, both conscious and subconscious ».

Henri Poincaré (1854 - 1912)

Chapter 1

Introduction

Contents

1.1 Sequential Break-down of Operations 1

1.1.1 Optimization Context 3

1.2 Decentralized Collaborative Editing 4

1.2.1 Collaborative Editing Context 6

1.2.2 Approach by Weiss et al. 7

1.2.3 Approach by Preguiça et al. 12

1.3 Contributions 15

1.4 Overview and structure 16

1.1 Sequential Break-down of Operations

The rapid growth in the use of modern computer technology has increased the demand for higher performance in all areas of computing. This demand for ever greater performance led to growth in hardware performance and architecture evolution that resulted in a stress on compiler technology and research communities. Since a high-performance microprocessor is at the heart of every general-purpose computer, from servers, to desktop and laptop PCs, to cell-phone platforms such as the iPhone. Therefore, increasing the performance is considered to be a permanent challenge in computer science. Since a 32 bit processor performs operations on 32 bits. Therefore, any transformation of data structure required decomposition in successive operations on 32 bits. A well-known technique used in exchanging contents of registers is to make copies of the initial data. However, it is possible that, dealing with large data structure, such technique may generates some memory errors and slow down the performance of computations. Moreover, to perform such operations, a number of registers in a micro-processor, through compiler or an electronic

circuit, are involved, and to complete such operations, it requires to make copies of contents of registers in the cache memory or in RAM or alternatively, it requires signal duplication in the chip design. In both cases, it causes, a loss of speed or an extra power consumption [BGT09].

> *"One of the possible approaches to this issue is the sequential break down of operations in such a way that it does not require any extra variable other than the variables available as input. Ultimately, it results in requiring no extra writing memory, therefore, such computations find applications in program optimization and chip design".*

Moreover, such *"In Situ Design of Computation (IDC)"* enables to improve performance of calculations and to calculate an operation related to n registers by a sequence of assignments using only these n registers [SCT].

Such *in situ* computation are applied in computing mappings from a set S^n to itself by performing a sequence of operations (assignments). Therefore, the mapping $E : S^n \longrightarrow S^n$ can be interpreted as the parallel computation of n assignment mappings of the form $S^n \longrightarrow S$. These mappings (n assignments) compute mapping E, either by mapping n input component variables onto n output component variables separately, or by modifying n component variables simultaneously. Such way of computation is termed as *"In Situ Design of Computation (IDC)"*. The idea of such type of computations is explained by the following preliminary example.

Example 1. *Let $E : \{0, 1\}^2 \longrightarrow \{0, 1\}^2$ be a mapping defined as $E(x_1, x_2) = (x_2, x_1)$ that actually, exchanges two boolean variables. A basic program that completes the required task is: $x_0 := x_1$, $x_1 := x_2$, $x_2 := x_0$, whereas, an in situ program that computes mapping E is $x_1 := f_1(x_1, x_2)$, $x_2 := f_2(x_1, x_2)$, $x_1 := g_1(x_1, x_2)$. The in situ program avoids using extra variable x_0, with $f_1(x_1, x_2) = f_2(x_1, x_2) = g_1(x_1, x_2) = x_1 \oplus x_2$.*

As further interpretations, the mapping $S^n \longrightarrow S^n$ (for $S = \{0, 1\}$) interprets boolean networks that have been vastly studied for their theoretical interest in computer science as for their potential applications in nature (genetic networks [Kum96], neuron networks, etc) or in the social sciences [Kel96]. In the context of chip design, such approaches (like *in situ*) are considered to be important and are discussed in electronic oriented papers (see for example [RAS97]). In (MINs) multistage interconnection networks [Agr83, BFJM89, Tom67], mappings $S^n \longrightarrow S$ (assignments) can be interpreted as set of edges that form bipartite graph between the set S^n and S^n. For all most 40 years, multistage interconnection networks (MINs) is considered to be an active research area. Many investigations exist about MINs considering various applications, for instance [Jur01, TM03, TH97, WT00, YW02, ZA02]. The technique for *in situ* computations can be viewed in the context of MINs,

due to the fact that, making successive modifications of consecutive components of a transformation in S^n is equivalent to routing a butterfly network when $S = \{0, 1\}$, or a generalized butterfly network with greater degree for an arbitrary finite set S [BGT09]. It has been proved that such *in situ* computation exists for linear mapping on binary field and some combinatorial results have also been discussed [Mir01].

1.1.1 Optimization Context

In recent years, many computer scientists investigated the area of optimization including compiler optimization, processor optimization, code optimization [BDGR06], algorithm optimization etc. The optimization of specific linear algebra problems has been discussed on a large scale because such type of optimizations have significant effect on processor performance. Whaley and Dongarra discuss optimizing the widely used Basic Linear Algebra Subroutines (BLAS) in [WD98]. Chatterjee et al. discuss layout optimizations for a suite of dense matrix kernels in [CJL+99]. Park et al. discuss dynamic data remapping to improve cache performance for the DFT in [PKBP00]. Frigo et al. in [FLP+99], discusses the cache performance of cache oblivious algorithms for the matrix transpose, FFT, and sorting. Optimizing blocked algorithms has been extensively studied [LRW91].

> A *"An important optimization, affecting the performance of compiler code, is register allocation. Register allocation has been studied extensively in compilation and is a NP-complete problem. In* 1981*, Chaitin et al. [Cha82, CAC+81] modeled the problem of assigning temporary variables to k machine registers as the problem of coloring, with k colors the interference graph associated to the variables".*

In general, register access is faster than memory access. Hence it is preferable to use register than memory whenever it is possible. When it is not possible to use register then some variables must be transferred to memory. This load/store operation has its cost. To avoid this cost, some classical approaches have been introduced like in graph coloring algorithms [BDGR06]. An iterated register coalescing algorithm, proposed by Appel and George [GA96] is a modified version of previous developments by Chaitin et al. [Cha82, CAC+81] and Briggs et al. [BCT94]. In these heuristics, spilling, coalescing (removing register-to-register moves), and coloring (assigning variables to registers) are done in the same framework. These techniques are very useful in compiler optimization but still have to be revised to get better results. Burckel et al. [BG08] proposed methods to compute mapping sequentially (by designing programs/electronic circuits), for performing operation on k registers of any sizes in a processor, in such a way that it does not requires extra working memory (such as other registers or external memories). These methods are directly connected with processor and compiler optimization in the sense of applications.

Moreover, it is proved that any mapping on $\{0, 1\}^n$ can be computed by such an *in situ* program and three types of assignments are sufficient to complete these operations [Bur96, BM00]. A boolean linear mapping of dimension n is computed with a sequence of $2n - 1$ linear assignments [BM04a, BM04b]. It is also proved that every mapping E on S^n can be computed by an *in situ* program of length $5n - 4$ whereas $4n - 3$ assignments are required when $|S|$ is a power of 2. Moreover, the maximal length of the program is $2n - 1$, if the mappings are bijective [BG08, BGT09].

1.2 Decentralized Collaborative Editing

Mobile technology and the ability to build connection between devices leads the extension of existing applications into new realms, so that mobile device users could communicate, process and present information. The rapid development of computer network inspired the advancement of real-time collaborative editing in which users are not bound to be in the same location, they can be at different sites that are geographically dispersed. Collaborative editing is a part of Computer Supported Collaborative Work (CSCW) [EGR91], and is a major research area in computer science for over two decades. Collaborative editing systems are used to allow physically dispersed people to edit a shared textual document [EG89, KP93, RNRGa96a, MO92, SJZ+98, SM96], to draw a shared graph structure [GM94, KBL93], to record ideas during a brainstorming meeting [HO92], or to hold a design meeting [OOSC92]. These documents could be articles, wiki pages and programming source code. A real time collaborative editor can be considered as a shared software application that makes it possible for several people to modify a shared document, provided they are connected to each other by any suitable network system.

Although being distributed applications, Real-time collaborative editors (RCE) are specific in the sense that they must consider human factors. So, they are characterized by the following requirements [Imi09]:

- High local responsiveness: the system has to be as responsive as its single-user editors [EG89, SJZ+98, SXS+06];
- High concurrency: the users must be able to concurrently and freely modify any part of the shared document at any time [EG89, SJZ+98];
- Consistency: the users must eventually be able to see a converged view of all copies [EG89, SJZ+98];
- Decentralized coordination: all concurrent updates must be synchronized in a decentralized fashion in order to avoid a single point of failure;
- Scalability: a group must be dynamic in the sense that users may join or leave the group at any time.

A real-time collaboration editor can be based on a centralized or a replicated architecture.

"In a centralized architecture, a central server is responsible to keep the shared document and to manage various aspects of the collaboration, i.e., the consistency, ordering of updates, resolving conflicts and the session membership. It requires that every user propagates his/her action to the central server, then the server will apply these changes to the document to ensure the intended document state [Cit07, CMR07] ".

"In a decentralized or replicated architecture, there is no central server to hold the shared document. Each participating site holds a copy of the shared document (replica) to work on separately. Each participant requires to broadcast its actions/updates to all participating sites so that each site can update their replicas accordingly. As compared to centralized architecture, decentralized architecture is more attractive specifically in wireless and ad-hoc networks. The absence of central server enables user to continue his/her work even in case of disconnection. The collaboration is managed by individual devices in the absence of central server [Cit07, CMR07]".

This issue presents challenges in implementing collaborative editors in a replicated architecture, especially in a mobile network which is characterized by limited resource reliability and availability. One of the main technical challenges in real time collaborative editing can be explained in the following example.

Example 2. *Suppose that two users U_1 and U_2 edit/modify a shared document that contains, word "Hear". Let user U_1 deletes character 'H', then inserts character 'N', so that word changes into "Near". Let user U_2, before he receives either edit from U_1, deletes character 'a' so that the original word changes into "Her". Both of the users U_1 and U_2 then receive updates that were applied to versions of the document that never existed on their own machines. Still it is a challenge to figure out how to apply edits from remote users, which were originally created in versions of the document that never existed locally, and which may conflict with the user's own local edits (explained with more detail in part II, chapter 7, section 7.1.6).*

Moreover, since users make conflicting modifications to their copies, these multiple copies of the same document can cause confusion. The conflicting modifications must be integrated into a coherent document. When users stop working on different versions of a document, *i.e*, following delivery delays, the problem of integration becomes even more complex. To address such issues, indexing the content of shared documents seems to be the basic requirement.

"one of the possible approaches is that the shared document be mapped to an interval $I = [a, b]$ (where, $0 \leq a < b$ and $a, b \in \mathbb{R}$) such that each character/line or object contained in the document is associated with a unique identifier in the interval I. For this purpose, we introduce a precision control indexing method that generates these identifiers such that the identifiers are real. We start in computing these identifiers using midpoint formula with particular modification and rounding conditions. It makes sure that each user is able to insert elements (character, line, etc) with different identifiers in the same interval and is able to knows his local as well as global cardinality corresponding to the set of identifiers".

1.2.1 Collaborative Editing Context

A comparison of several approaches to the problem of collaboratively editing shared text has been proposed by Ignat et al. [IOM+07]. A number of collaborative applications are based on operational transformation (OT) [EG89, LL04a, LL04b, SE98, SJZ+98] but OT considers collaborative editing based on non-commutative operations and it transforms the arguments of remote operations to take into account the effects of concurrent executions. In fact, OT needs two conditions of correctness [RNRä96]. The transformation must enables concurrent operations so that the execution should be in order, and the transformation functions themselves should commute. The second condition is more complex than the first one [PMSL09]. It has proven that all previously proposed transformations don't ensure these properties [IMOR03, IROM06]. Moreover, the solutions proposed later [LL07, OMUI06, SS06] are complex, and their correctness can not be easily verified.

"All the OT-based algorithms [EG89, LL07, Ost06, RNRGA96b, SJZ+98] belong to the logical level approach. The logical-level approach needs to maintain and scan historical log to decide right transformation paths for remote update operations. The transformation procedure becomes expensive under heavy workloads. Furthermore, operations defined on the logical view lose their correct position indexes as the document is edited, which creates difficulties for undo operations. OT-algorithm [Ost06] also introduces "tomb stone" as part of their data structure. But the purpose is to resolve transforming ambiguities on operations that update the same portion of a document.".

Oster et al. [OUMI06] proposed WOOT algorithm to manage cooperative editing and to support insert and delete operations . The proposed algorithm WOOT, is also identified as a CRDT (Commutative Replicated Data Types) [PMSL09]. In WOOT, each character assigned a unique identifier, and maintains the identifiers of the previous and following characters at the initial execution time. Moreover, data structure grows indefinitely, due to the reason that there is no garbage collection or restructuring [SPBZ11].

The WOOT [OUMI06] algorithm and the technique used in the editor TeNDaX [LHWBD06] belongs to the physical-level approach. The WOOT algorithm defines a pair (site identifier, local counter) to create a unique identifier for each character. The site identifier is unique to each site. The local counter is incremented each time a new operation is executed. In TeNDax, each character is assigned a unique integral identifier by a central server. In both approaches, character identifiers are indexed (such as hash-tables or B-trees) so that they can be quickly searched to determine data dependencies between operations. The major problem in their approaches is that the identifiers do not carry ordering information. Therefore, characters that are logically consecutive may not be stored physically consecutive on disk [QC09].

Recently, *Weiss et al. proposed the Logoot CRDT [WUM09] where they introduce identifiers as tuple* $\langle i, s, c \rangle$ *with 'i' a digit, 's' a peer identifier and 'c' a clock. While, an approach consists of Treedoc's dense binary tree is proposed by Nuno et al. [PMSL09]. Weiss et al. introduced a position identifier as a list of (long) unique identifiers, and Logoot does not flatten. As compared to Treedoc technique, Logoot has a high overhead. Another technique to ensure consistency is executing operations in the same order at all copy (replicas) [Lam78]. To make sure that an edit position has the same meaning at all replicas needs either operating replicas in lock-step, or operational transformation [SE98]. In the Treedoc design proposed by Nuno et al. [PMSL09], common edit operations execute optimistically, without a latency and replicas synchronize only in the background.*

However, we noticed that both of the algorithms [PMSL09, WUM09] for generating unique line and position identifiers do not support some critical situations. we discuss these problems by quoting examples, constructed by using original algorithms.

1.2.2 Approach by Weiss et al.

In this section, we describe, in detail, the algorithm for generating identifiers, proposed by Weiss et al. [WUM10]. The algorithm (proposed by Weiss et al.) consists in two parts, Function 1 and Function 2. The authors, consider an edit of a shared document as a set of operations, called patch. So performing a set of operations (patch) results in insertion of lines that are contiguous or grouped by contiguous blocks. In Function 1, N presents the size of a block. So between two lines indexed by two line identifiers p and q, N denotes the number of identifiers, that are to be generated on site s.

The boundary use to limit the distance between two successive line identifiers so that the constructed line identifiers can be apportioned. Therefore, boundary helps in grouping line identifiers so that the space can be created for

next subsequent insertion. Within chosen boundary the identifiers can be apportioned randomly and is called random strategy. The function $prefix(p, i)$

Function 1:(How to generate new Line Identifier)

```
   Function: generateLineId
   Input: (p, q, N, boundary, site)
1  begin
2      let list := π
3      index := 0
4      interval := 0
5      while (interval < N) do
6          index++
7          interval := prefix(q, index) − prefix(p, index) − 1
8      end
9      Let step := min(interval/N, boundary)
10     r := prefix(p, index)
11     for j:=1 to N do
12         list.add(constructId(r + Random(1, step), p, q, site))
13         r :=r + step
14     end
15     return list
16 end
```

(see line 10, Function 1) returns a number in base-BASE. The second part

Function 2:(How to Construct new ID)

```
   Function: constructId
   Input: (r, p, q, site)
1  begin
2      let id:={}
3      for i := 1 to |r| do
4          d:=ith digit of r
5          if d = p_i.digit then
6              s := p_i.siteid
7              c := p_i.clock
8              else if d = q_i.digit then
9                  s := q_i.siteid
10                 c := q_i.clock
11                 else
12                     s := site.identifier
13                     c := site.clock + +
14                 end
15             end
16         end
17         id := id.⟨d, s, c⟩
18         return id
19     end
20 end
```

of the algorithm [WUM10] (Function 2) constructs identifiers and is executed through Function 1. In fact, Function 1 finds a place for N lines between the shortest possible prefixes of identifiers p and q. After selecting the shortest possible prefixes for N, Function 2 returns N identifiers.

We analyse capabilities of Functions (1 and 2) in the context of identifiers exchange scenario which is explained below.

Identifiers Exchange Scenario

For the sake of simplicity, we take two users U_1 and U_2 at *site* 1 and *site* 2 (see Figure 1.1) and empty document with initial and final identifiers. Suppose that

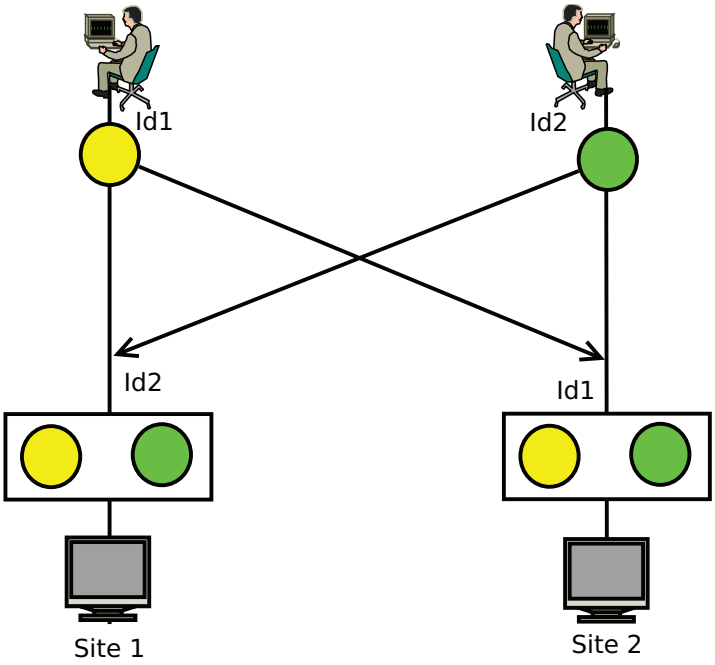


Figure 1.1: Identifiers exchange scenario

two identifiers are generated at *site* 1 and *site* 2 respectively corresponding to the insertion of two characters "a" and "b".

Suppose that updates are executed and identifiers are exchanged such that each user sees two identifiers. Our investigation is that, if any of two users inserts a new character between "a" and "b" then new identifier lies between identifiers corresponding to characters "a" and "b" or not? Let us observe "Approach by Weiss et al." under identifier exchange scenario. According to this approach, for empty document, we take the initial and final identifiers $\langle 0,\, NA,\, NA\rangle$ and $\langle MAX,\, NA,\, NA\rangle$ respectively, where $MAX = BASE - 1$. After exchanging identifiers by two users, each user can have some of the identifiers of the form

$$\langle i,\, s_j,\, c_j\rangle,\ \text{for } j = 1,\, 2, \ldots,\ \text{and } i = 0,\, 1, \ldots, MAX$$

Suppose that these two identifiers are

$$\langle i,\, s_1,\, c_1\rangle \text{ and } \langle i,\, s_2,\, c_2\rangle$$

$$s_1 < s_2 \text{ or } s_1 = s_2 \text{ and } c_1 < c_2$$

Suppose that user U_1 intends to insert new character between the twos that are already there. We detect some drawbacks in Algorithm (consists of functions 1 and 2) that are given below.

Infinite Iteration: For two identifiers with same values of i, Algorithm (consists of Function 1 and Function 2) does not generate any new identifier due to the fact that the value "$interval = -1$" (see line 7, Function 1). It runs the loop for an infinite iteration.

Now, suppose that we exit from infinite iteration by defining some extra conditions for the Algorithm then still there is order preservation problem as discussed below.

Order Alteration: To insert one identifier, we can take the value $N = 1$. Since selecting value for boundary is optional so it can be taken as "$boundary = 1$" then for two identifiers

$$p = \langle 0,\ s,\ c\rangle \quad \text{and} \quad q = \langle 0,\ s,\ c\rangle$$

with $p < q$ (say), the algorithm returns values as given below

$$\text{step} = min(1,\ 1) = 1 \ \text{(see line 9, Function 1)}$$

and

$$\text{prefix}(p,\ index) = 0 \Longrightarrow \text{r} = 0 \ \text{(see line 10, Function 1)}$$

After execution of Function 1 for these values, Function 2 returns new identifier as

$$id = \langle 1,\ s,\ c\rangle \ \text{(see line 17, Function 2)}$$

To preserve our desired order, computed identifiers should follow

$$p = \langle 0,\ s,\ c\rangle < \langle 1,\ s,\ c\rangle < \langle 0,\ s,\ c\rangle = q$$

which is not true because $1 \notin [0,\ 0]$.

Formula failure: The formula proposed (by Weiss et al.) to compute number of identifiers between two identifiers does not support some cases. For example, suppose two lines are identified by the identifiers

$$p = \langle k_1,\ 4,\ 7\rangle\langle k_2,\ 9,\ 5\rangle \ \text{ and } \ q = \langle k_3,\ 5,\ 3\rangle\langle k_4,\ 3,\ 6\rangle\langle k_5,\ 3,\ 9\rangle$$

with assumption that $BASE = 100$, $boundary = 10$ and

$$\text{Suppose that } k_1 = k_2 = k_3 = k_4 = k_5 = k$$

Suppose that we are interested to insert N lines between two lines identified by the identifiers p and q, then to compute number of identifiers using the formula we proceed as follows

$$\text{prefix}(q,\,1) - \text{prefix}(p,\,1) - 1 = \begin{cases} -1 & \text{if } k_3 = k_1 \\ 0 & \text{if } k_3 = k_1 + 1 \end{cases}$$

Thus, formula does not work in this case, moreover, it does not work to compute cardinality of the identifiers for sizes 2, 3 and so on, for the values as prescribed above.

Redundancy of Identifiers: We explain redundancy of identifiers by proposing the following scenario (see figure 1.2). Suppose that two users at two different sites *site* 1 and *site* 2 have an empty shared document (see figure 1.2) with initial and final points

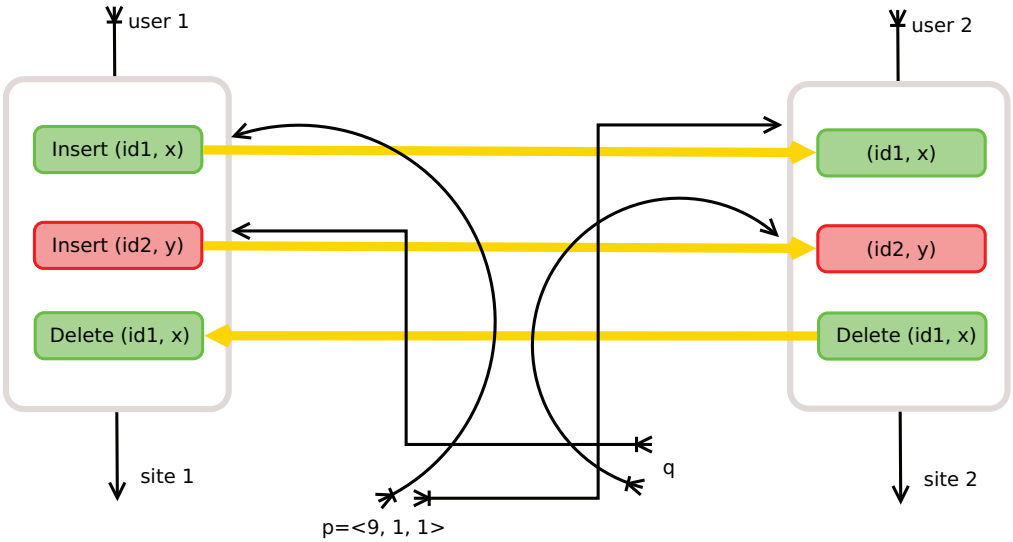


Figure 1.2: Redundancy of Identifiers

$$lb =< 0,\, NA,\, NA > \quad \text{and} \quad le =< 10,\, NA,\, NA >$$

respectively.

The operations labelled in the figure 1.2 perform the following tasks.

1. Insert (id_1, x), inserts character "x" with an identifier "id_1".
2. Delete (id_1, x), deletes character "x" with an identifier "id_1".
3. Insert (id_2, y), inserts character "y" with an identifier "id_2".

Let first user (*user 1*, with assumptions, $s = 1$ and initial clock $c = 0$) inserts a character "x" (we take $N = 1$, for insertion of single character)

and an identifier p (say) is created that, according to the algorithm (consists of Function 1, Function 2), takes the value $p =< 9, 1, 1 >$ (see line 4 to line 17, Function 2). We get the value $p =< 9, 1, 1 >$ for the reasons that any value can be chosen for boundary, therefore, it is possible that (see for details, line 4 to line 9, Function 1)

$$\text{boundary} \geq \text{interval/N}$$

In this case the following may holds

$$\text{step=min(interval/N, boundary)=interval/N}$$

It permits that

$$\text{for } N = 1, \text{ step=9}$$

Thus, algorithm will return identifier $p =< 9, 1, 1 >$ such that

$$\{lb, p, le\} = \{< 0, NA, NA >, < 9, 1, 1 >, < 10, NA, NA >\}$$

Let *user 1* inserts another character "y" after "x" and an identifier q (*say*) between

$$p =< 9, 1, 1 > \quad \text{and} \quad le =< 10, NA, NA >$$

is created. We observed that, this new identifier q generated by algorithm (Function 1, Function 2) could be equal to p or le. Let $q = p$, but since the contents of p and q are not necessarily the same. Therefore, let the content of p be "x" and content of q be "y". The updates are sent to *user 2* at *site* 2 and suppose that *user 2* intends to perform operation "Delete (id_1, x)" and send update to *user 1*. The execution of "Delete" operation at *site* 2 (because of redundant identifiers) does not guarantee the desired result and may leads to divergence.

1.2.3 Approach by Preguiça et al.

Nuno preguiça et al. [PMSL09] proposed an approach to construct unique position identifiers in which they introduce unique disambiguators. They called this approach as $UDIS$ approach. The proposed algorithm (Function 3) is described as below. We analyse Function 3 also, under "Identifiers Exchange Scenario" described in Section 1.2.2 (Figure 1.1).

Function 3:(Generating POSID)

Result: newPosID ($PosID_p$, $PosID_f$)

// d: new disambiguator.;

Require: $PosID_p < PosID_f \ \wedge \nexists$ atom x such that $PosID_p < PosID_x < PosID_f$;

begin

  if $PosID_p /^+ PosID_f$ **then**

    Let $PosID_f = c_1 \odot \ldots \odot (p_n : u_n)$;

  end

  return $c_1 \odot \ldots \odot p_n \odot (0 : d)$;

  else if $PosID_f /^+ PosID_p$ **then**

    Let $PosID_p = c_1 \odot \ldots \odot (p_n : u_n)$;

  end

  return $c_1 \odot \ldots \odot p_n \odot (1 : d)$;

  else if $MiniSibling(PosID_p, PosID_f \vee \exists PosID_m > PosID_p$: $MiniSibling(PosID_p, PosID_m \wedge PosID_m /^+ PosID_f$ **then**

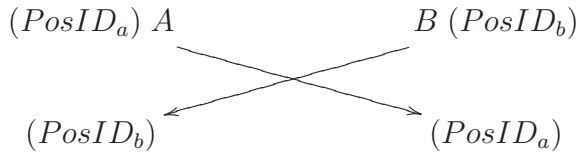
    Let $PosID_p = c_1 \odot \ldots \odot (p_n : u_n)$;

  end

  return $PosID_p \odot (1 : d)$

end

Order Alteration Suppose that an empty document is marked with initial and final identifiers (0, d_0), and (1, d_1) respectively. Let two users A and B edit/modify the document and exchange the corresponding identifiers (for example, $PosID_a$, $PosID_b$) by executing updates.



After performing updates and exchange of identifiers, each user will see two identifiers. Now, suppose that user B intends to create new identifier between two identifiers (already computed, one of them is sent by other user). Let us compute this new identifier according to Function 3 and explain the problem with more detail using Figure 1.3.

Figure 1.3 presents an empty shared document with initial and final point marked as 'Beg' and 'End' respectively. Now any new position identifier must be the right child of the node 'Beg' or the left child of the node 'End'. Let user A wants to insert a character "a" and user B wants to insert a character "c" at the same position. Suppose that dA and dB

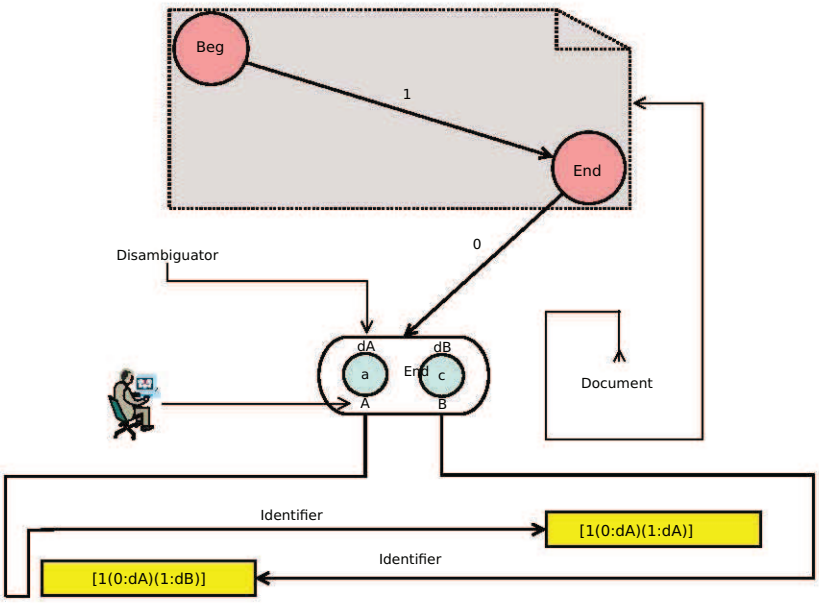


Figure 1.3: Generating POSID

denote the disambiguators for users A and B such that $dA < dB$ then according to the first "if condition" (see lines 4 to 6) of Function 3

$$PosID_a = [1(0 : dA)] \text{ and } PosID_c = [1(0 : dB)]$$

Now two nodes marked as "a" and "c" are the siblings. Let user B is interested in insertion of a new character "a_1" between "a" and "c" then he/she/it must has to create new identifier according to "if condition" (see lines 12 to 15) of Function 3, that returns

$$PosID_{a_1} = [1(0 : dA)(1 : dB)]$$

Let

$$dA = (counter, siteID) = (1, 1)$$

and

$$dB = (counter, siteID) = (1, 2)$$

then

$$PosID_a = [1(0 : (1, 1))]$$

and

$$PosID_c = [1(0 : (1, 2))]$$

Now, identifier for character "a_1" is computed as

$$PosID_{a_1} = [1(0 : (1, 1))(1 : (1, 2))]$$

Following the definition as prescribed by Nuno preguiça et al. [PMSL09], it gives

$$PosID_a < PosID_{a_1} \text{ and } PosID_c < PosID_{a_1}$$

$$i.e.,\ PosID_{a_1} \notin [PosID_a,\ PosID_c]$$

We focus to deal with such kind of problems that pose restriction to the users, we intend to provide an environment where the identifiers follow a strict order and can be seen like an ordered set. It will provide an open option for the participants to start, whereas, the set of identifiers will be a finite ordered set that preserves a strict order.

1.3 Contributions

We aim to extend the idea of sequential break down of operations applied in computing mappings from a set S^n to itself by sequence of assignments that does not require any extra variable other than the variables available as input variables. Such computations contribute in improving the performance of calculations and to calculate operations related to n (say) registers through sequence of assignments that involves the same number of registers. Such type of contributions provide motivations and inspired us to look forward in this area of research. The existence of such *in situ* computations over fields, rings and for the boolean case has already been proved [Bur07, BGT09].

We start in explaining the existence of these computations over fields and rings with different examples to provide a strong foundation. Then, we investigate the possibility of computing inverse mapping E^{-1} (say) by sequence of assignments (keeping coefficients in fields or rings) generated by corresponding sequence of assignments that computes mapping E and provide counter examples. For the fields, we propose an alternate simple technique using Bézout's Identity. For the rings, we design algorithm and implemented the idea successfully. We prove the existence of such computation over the set of integers and investigated bound over the minimum number of assignments (required to compute mappings) by developing relations between mappings keeping Fibonacci numbers as coefficients. We established relation between determinant of matrix (that expresses coefficients of corresponding mapping) and the coefficients of assignments that combine to compute mappings sequentially. This identity provides quick reply in investigating the sequence that can compute inverse mappings. In boolean case, it has already noticed that, every assignment x_i (say) performed in an *in situ* program to compute a boolean bijective mapping must be linear in x_i [BGT09]. Based on this linearity property, we extend investigation and proposed a technique to compute bijective mapping with respect to set of its components (boolean mappings) through construction of boolean polynomials.

`In contributing to Collaborative Editing Systems.`

We aim to design a decentralized collaborative editing system (DCES), focusing into the existing limitations and to contribute in the required perfections so that the system could be able to deal with the modern and newly launched applications based on the collaborative editing. Some of the issues (*e.g*, high concurrency, correct integration of updates) require proper indexing of characters/lines or objects that comprise a shared document. For proper indexing, it needs to create identifiers for each character/line or object. Some techniques are available to generate these identifiers, see for instance [PMSL09, WUM10]. These techniques avoid to use real numbers due to the precision issue and the proposed algorithms don't deal all cases of preserving order for these identifiers, specifically during the remote computation and exchange of points.

"We introduce a precision control indexing method to generate a set of unique identifiers such that these identifiers can be used for indexing characters/lines or objects. These identifiers are still real numbers with a specific controlled pattern of precision. The set of these identifiers is finite that enables us to compute local as well as global cardinality. To generate a new identifier, only the information about two neighboring identifiers is required. From an identifier, the site, where it has been generated, can be identified immediately. We start in indexing shared document, initially, with an interval $I = [a, b]$ *with* $0 \leq a < b$ *for* $a, b \in \mathbb{R}$ *and new identifiers lie within the interval* I*. We suppose a collaborative editing system as a network* $\aleph$ *of* n $(n \in \mathbb{N})$ *users/sites or peers such that each site/peer also assigned an identifier generated under specific precision. We implement the idea successfully by designing algorithm that guarantees order preservation for the set of identifiers as well as over the subsets ".*

1.4 Overview and structure

An overview of this thesis is outlined below.

Part I illustrates the existing strategies and contributes in developing new strategies for sequential break-down of operations.

- ⋆ Chapter 2, gives a quick overview about compiler/processor optimization and provides motivations in the area of research under discussion.
- ⋆ Chapter 3, starts in providing the basic concept of *"In Situ Design of Computation (IDC)"* for mappings, explains its existence over fields, gives alternate strategy and illustrates with examples.
- ⋆ Chapter 4, verifies the existence of *"IDC"* for mappings over rings, explains and discuss the case of inverse linear mappings.

- ⋆ Chapter 5, proves the existence of *"IDC"* for mappings over integers, develops relations to determine bound over the number of assignments and makes connection between matrices (presenting coefficients of mappings) and the coefficients of assignments involve in computing mappings.
- ⋆ Chapter 6, illustrates in more simple ways *"IDC"* for bijective and general boolean mappings and provides foundation to develop *"IDC"* through polynomials over GF(2).

Part II provides foundation to design decentralized collaborative editing system based on precision control indexing method.

- ⋆ Chapter 7, presents fundamental concepts related to collaborative editing systems and provides a model for decentralized collaborative editing system.
- ⋆ Chapter 8, provides precision control indexing strategy, illustrates its implementation in generating identifiers used for indexation in collaborative editing systems, proves the uniqueness of identifiers and provides method to compute cardinality of these identifiers.
- ⋆ Chapter 9, presents some important properties of identifiers generated under precision control strategy, analysis of the algorithm that generates identifiers, and comparisons of the output of the strategy under certain conditions.
- ⋆ Chapter 10, provides different models for exchanging points across the network and computing new points based on remote points, and estimates the effect on cardinality of identifiers through experimentations.

Part III Presents an evaluation.

- ⋆ Chapter 11, describes future research directions and presents conclusion.

Annexes 11.2 .

1. Annex A, provides an extensive experiment that helps in observing the effect on cardinality during the exchange of points (see for detail, chapter 10).
2. Annex B, provides Maple algorithm that implements the idea of *in situ* computation over rings (see for detail, chapter 4).

Part I

Sequential Break-Down of Operations

Chapter 2

Compiler, Processor Optimization (A quick overview)

Contents

2.1 Compilers 22

2.1.1 Compiler Optimization 23

2.2 Processors 24

2.2.1 Processor Optimization 26

2.3 Applications 26

2.4 Motivations and Objectives 27

Relatively, in a short time, computers evolved from huge mainframes to small and elegant computers including ultra-portable hand-held devices. With passage of time the size of computers became smaller but computers became faster. The first commercial computer (UNIVAC I), occupied 943 cubic feet space and was able to perform $1,905$ operations per second. Now, a processor equipped in an electronic shaver is able to perform the basic job. A processor interprets sequences of particular instructions, while a computer program is written using a high-level programming language. The program text must be converted into suitable sequence of instructions so that it can be processed by processor, this task is performed by compilers.

In this chapter, we take a brief look at the basic concepts of compilers, processors and their applications. Section 2.1 discusses compilers, compiler optimization and the techniques of compiler optimization. Section 2.2 describes processors and processor optimization. Applications are described in section 2.3, whereas, section 2.4 gives motivations and objectives.

2.1 Compilers

The main purpose to execute relatively simple commands is, to reduce the complexity of building and designing computers. A program readable for a computer must be developed by combining such kind of very simple commands into a program in what is called machine language. Machine languages are the languages that computers understand but are almost impossible for humans to use because they are entirely based on numbers. Therefore, programmers, use either a high level programming language or an assembly language. High level languages (for example, C++, Java, etc.) could be very different from the language that a computer can execute. A high level language must be converted to a language that a processor can understand, therefore, some measures are required to bridge the gap and compiler is actually the mean to do this task. Compilers were introduced in the early $1950's$ [ABE+97, CD97, Ghu, gM10, Wir96].

A compiler (more generally, translator) is a software application that translates (compiles, converts) a program (code) written in a high-level programming language that is suitable for human programmers into the low-level machine language that is required by computers. In addition, compilers also point out the obvious programmer's mistakes. A cross compiler is the compiler that generates code for a computer different from the one that executes the compiler.

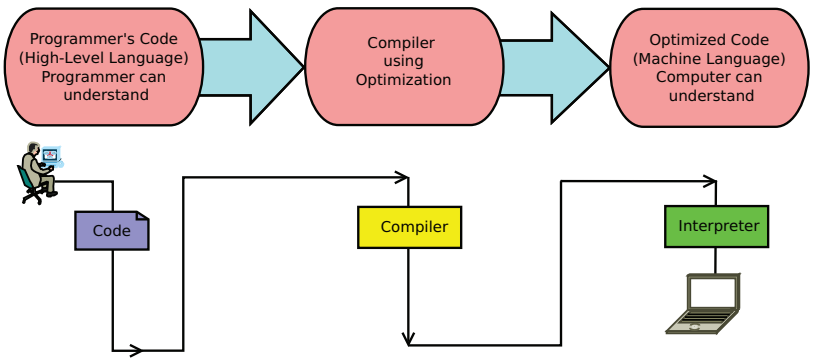


Figure 2.1: Compiler Interaction

Many compilers are available for the same language, for instance, a list of FORTRAN compilers is shown in the Table 2.1. Besides Table 2.1, there is a number of compilers. For example, C Compilers, C++ Compilers, ALGOL Compilers, Haskell Compilers etc. The first compiler was constructed for the language Fortran (formula translator) around 1956, the success was considered as a daring step because it was not sure to achieve the goal. Almost 18 years

Compiler	*Author*	*Windows*	*Unix-like*
Absoft Pro Fortran	Absoft	Yes	Linux and Mac OS X
FTN95	Silverfrost	Yes	No
G95	Andy Vaught	Yes	Yes
gfortran	GNU	Yes	Yes
Intel Fortran Compiler	Intel	Yes	Linux and Mac OS X
Lathey Fortran	Lathey Computer Systems, Inc.	yes	Linux only
NAG Fortran Compiler	Numerical Algorithm Group	Yes	Yes
Open64	Google, HP, Intel, Nvidia, PathScale, Tsinghua University and others	Yes	Yes

Table 2.1: Fortran Compilers

of effort were involved in developing the first compiler, that is why the project was remarked as one of the largest programming project of the time.

Selecting a well defined and well structured source language helps in improving the quality and reducing the complexity of the translation process. Algol 60, first time, has introduced the technical foundations of compiler design.

Now, the translation process is guided by the structure of the analyzed text. According to the syntax, the text is decomposed into its components. However, the meaning of the source text is preserved by the translation.

The output of two different pieces of code in assembly language could be equivalent, but they may perform the task using different sequence of steps. As an example, there are different ways to add three numbers 1, 2 and 3, the computer could execute this. One of the possible ways would be to add 1 and 2 together and then add 3 to that result $(1 + 2) + 3$. Another way to add the three numbers would be to add 2 and 3 together, and then add 1 to that result $(2 + 3) + 1$. A compiler has options in which specific implementation of assembly language it will choose in making the translation from the high-level programming language [Mir01, Wir96].

2.1.1 Compiler Optimization

Generally, the optimization aims at finding set of instructions that make a particular pattern (sequence of instructions) so that this pattern can be replaced by another set of instructions. Compiler optimization enables computer programs to be more efficient and ultimately the whole process helps to increase the speed for compilation. In addition, it aims to minimize the usage of memory storage and power consumption.

Usually three types of optimizations are considered:

1. Global optimization: It seeks to reorder the sequencing of a program in order to eliminate redundant computations (for example, moving invariant operations outside loop bodies, collapsing loops, etc.).

2. Register optimization: It adjusts the allocation of machine registers to variables and intermediate quantities in such a way as to minimize the number of a register has to be stored and later reloaded.

3. Local (time) optimization: It seeks to adapt the code to exploit particular features of the machine architecture and to remove local mishandling such as loading a register with a value that it already contains.

Compiler Optimization Techniques

Compiler optimization techniques are categorized as, machine dependent, architecture dependent and independent. Machine dependent techniques describe the instruction level sensitivities of a compiler. Architecture dependent techniques describe the parts of programs that relate to the general hardware implementation, but not to a specific machine. Architecture independent techniques describe the aspects of program formulation that do not depend on a particular computer system or even on a type of implementation (e.g. pipeline processing). Until now, manufacturers focus on machine dependent techniques.

Programming techniques take advantage of the optimizing compilers and the system architecture, *e.g.*, BLAS (a library of Basic Linear Algebra Subroutines). The subroutines included in this library are able to provide significant enhancement in the performance of a program that is numerically intensive. ESSL (Engineering Scientific Subroutine Library) is an extension of BLAS library and includes high-performance mathematical routines for chemistry, engineering, and physics. Several run-time techniques have been introduced by computer architects based on the hardware. This enabled the processor that it can run any ready instruction from an instruction window [Cen, Kel96, Kum96, MO56, Sch73, Tom67].

2.2 Processors

Processor, also known as microprocessor (designed for microcomputers and micro controllers) or CPU (Central Processing Unit), is a complete computation engine that is integrated in a single chip. In fact, it is an integrated circuit, containing the arithmetic, logic, and control circuitry, and is used to interpret and execute instructions from a computer program. This integrated circuit, in combination with other integrated circuits that provide memory to store and execute the program, form a chip. Microprocessor registers used to hold temporary results, when the computation is being performed. Since these registers and microprocessors are made by the same technology, therefore there

is no speed disparity between them. Moreover, these registers act as processor memory.

To improve the performance of microprocessor a small memory has been introduced between microprocessor and main memory. This small memory is called cache memory, it is expensive but fast and is first time introduced in IBM 360/85 computer. The first microprocessor was the Intel 4004, introduced

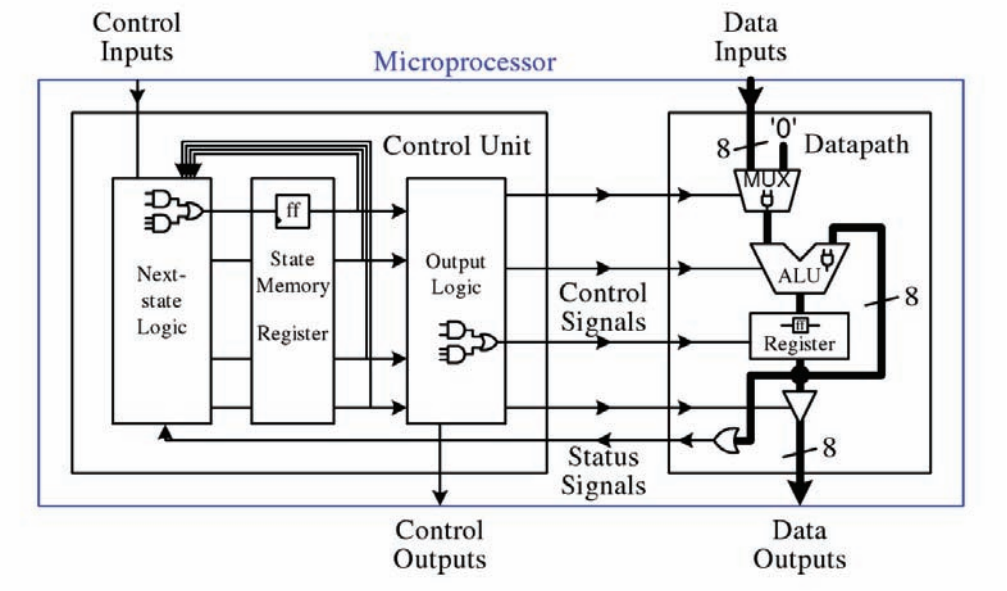


Figure 2.2: Microprocessor

in 1971. It was able to perform only subtraction and addition up to 4 bits at a time but everything was first time on a single chip. The microprocessors can be classified based on:

- the semiconductor technology of their design, (*e.g.*, TTL, CMOS or ECL).
- the width of the data format (4, 8, 16, 32, 64-bit) they process.
- their instruction set, (*e.g.*, CISC or RISC).

Due to low power consumption, CMOS (complementary-metal-oxide semiconductor) technology is preferred to use in portable computers and in other devices that use batteries while TTL (transistor-transistor logic) is commonly used. When high performance is needed, the ECL (emitter-coupled logic) is used. Older high-end mainframe computers, like the Enterprise System/9000 use ECL but current IBM mainframes use CMOS [BEM+92]. Four-bit devices are good for simple control applications and they are not so costly. CISC,

(complex-instruction-set computer) processors, which have 70 to several hundred instructions, are easier to program than RISC, (reduced-instruction-set computer) processors, but are slower and more expensive [ACP+08, IR87, SRAC91, Sla92].

2.2.1 Processor Optimization

How efficiently and effectively the processor executes instructions (provided in the form of a program designed by using some high level language) is determined by its internal design, also called its architecture. A processor can be considered as a combination of small blocks that organize to make a system. To optimize the design space, a model is required that can predict the performance of a processor as a function of the delays of the underlying blocks. With such a model, one can evaluate how a change in the delay of a given module will affect the system's performance and can use this information to optimize a design. Because the design space is complicated, therefore, it may be difficult to know how changing the delay of a module will affect the overall performance of a processor [ACP+08].

Processor architects continue their efforts to improve the performance of processor every year. Some of the major techniques used by processor architects are the use of wider data buses and registers, floating point units, pipe lining and super scale architecture. As processor speed continues to increase faster than memory speed, optimization to use the memory hierarchy efficiently become ever more important. Blocking [GL89] or tiling [Wol89] is a well-known technique that improves the data locality of numerical algorithms. The improvement obtained from tiling can be far greater. Tiling can be used for different levels of memory hierarchy such as physical memory, caches and registers. Multi-level tiling can be used to achieve locality in multiple levels of the memory hierarchy simultaneously [AS78, JDD90, KGS87, MSLW91, MC69, WM91].

2.3 Applications

A fast computer program (as a result of compiler optimization) is not only useful for computer scientist and computer architecture, it affects the general public as well. Compiler optimization helps to increase the efficiency and capabilities of not only the sophisticated software but also increase the demand of newly introduced computer based devices. Ultimately, compiler optimization directly affect the computer based technology used in particular as well as in common life. For example, an improvement in computer programs for the medical community can affect all communities. An improved resolution of images obtained through scanning process has a direct affect on doctors and patients. An optimization may turn life of the public to an ease. A number

of soft wares are being used only for the purpose of research to observe the output of different results. A maple program can take more than a week to produce the output of a program, but if the same program yields the result within one hour or less, imagine, how fast will be the conclusions based on these results.

Microprocessors are the result of the semiconductor industry's ability to place an ever-greater number of transistors in a single integrated circuit. Optimized processors at the heart of mobile products enable communications, computing, and multimedia functions to be efficiently executed. Microprocessors also play supporting role within larger computers as smart controllers for graphics displays, storage devices, and high-speed printers.

A vast majority of microprocessors are used to control everything from consumer appliances to smart weapons. Microprocessors are the main reason to have inexpensive hand-held electronic calculators, digital wristwatches, and the electronic games. They are also used to control consumer electronic devices, such as the programmable microwave oven and DVD player, to regulate gasoline consumption and anti lock brakes in auto mobiles, to monitor alarm systems, and to operate automatic tracking and targeting systems in aircraft, tanks, and missiles and to control radar arrays that track and identify aircraft, among other defense applications. Intel®Core™ i7 processor is the most ever advanced desktop processor, introduced by Intel corporation recently in 2008. The Core i7 processor (as shown in the Figure 2.3) is the first member of a new family of Nehalem processor designs and is the most sophisticated ever built, with new technologies that boost performance on demand and maximize data throughput. The Core i7 processor speeds video editing, immersion games and other popular Internet and computer activities by up to 40 percent without increasing power consumption.



Figure 2.3: Intel®Core™ i7 processor

2.4 Motivations and Objectives

Multi-core processors bring an evolution in computing technology. Almost, all modern computers are equipped with multi-core processors. Multi-core processors offer performance and productivity benefits beyond the capabilities of

today's single-core processors. This new trend of multi-core processors demanding new modifications in compilers so that they could be able to deal with new challenges. Modern compilers are introduced with a significant number of optimization that have different effect on quality and size of code, time taken and energy consumption etc. The code of the program that is to be compiled influenced the optimization as well and code optimization introduced new trend both in hardwares and softwares [HE08, Sar08, Sar09].

Current architectures contain one or more processors that are equipped with relatively small number of registers. The number of registers available on a processor and the operations that can be performed using those registers have significant impact on the quality of code generated by optimizing compilers. These registers are constantly requested in the operations, however, it is advisable to minimize the use of registers. It is proved that it is possible to calculate an operation related to n registers by a sequence of assignments using only these n registers. Moreover, if this operation is linear or bijective, the number of assignments is at most $2n$. In the general case, this number of assignments is at most $4n$.

It is a fact that current computer architectures reach their theoretical limits of performance. However, it is still possible to gain performance of calculations by performing calculations in a new way. Such type of calculations, actually, generalize the traditional principle of the exchange of two numbers A, B by the sequence:

$$\begin{aligned} A &:= A + B \\ B &:= A - B \\ A &:= A - B \end{aligned}$$

We aim to study these methods and focus to improve these techniques of computations, especially to develop methods and efficient heuristic algorithms to find these decompositions and implement these methods. We are interested in improving the bounds over these computations, to obtain new methods of computations for particular cases and implementing these methods in a compiler with an aim of optimizing code in machine language. This implementation can be provided in hardware through the design of new processors, in the software, by optimizing compilers upstream (pre-level language compilers) and downstream (post-machine language compilers).

Chapter 3

In Situ Design of Computation for Linear Mappings over Fields

Contents

3.1 Sequential Computation 30

3.1.1 *In Situ* Design of Computation (IDC) 31

3.2 Existence of IDC for Linear Mappings over Fields 33

3.2.1 Case-I 35

3.2.2 Case-II 38

3.2.3 Computing Inverse Mapping 39

3.2.4 Case-III 39

3.2.5 Possibility of Computing Inverse Mappings . . . 41

3.3 An approach using Bézout's Identity 41

3.3.1 Case-I 43

3.3.2 Case-II 43

3.3.3 Computing Inverse Mappings: 45

Sequential computation of linear mappings in such a way that it requires no extra variables other than the variables available as input, is an approach towards optimization (for instance, processor, compiler, memory, code). A fundamental step in this approach is investigating the existence of such computations over fields, *i.e.*, whether the coefficients of linear assignments (involve in computing the given mapping) belong to a field or not.

In this chapter, we aim to discuss in detail, the existence of such computations over fields (has already been proved by Burckel et al [Bur07]). We illustrate the corresponding assertion in section 3.2 and provides examples

that explain, how to construct a sequence of linear assignments that compute the given mapping and how to construct a sequence of assignments that compute inverse mapping. We investigate the possibility of computing inverse mapping and provide counter example. In section 3.3, we introduce an alternative approach using Bézout's Identity, and give examples that explain this approach.

3.1 Sequential Computation

Conversion of input into output under well defined sequence of basic computational steps leads towards the theory of sequential computations. A sequential program implements mathematical function that maps a set of inputs to a set of outputs. These mathematical functions are well defined and the notion of computable functions has been introduced earlier by Church, Kleene and Turing. These functions are frequently used in untyped lambda calculus, recursive functions, and Turing machines [Chu36, Kle36, Tur37].
These basic models help in designing and reasoning for programming languages, domain theory and denotational semantics introduced by Scott and Strachey, and provide a global mathematical setting for sequential computation, building on top of the foundational theories [SS71]. This advancement interconnects different programming languages and makes connection with the mathematical world of algebra, topology, and logic. It inspires the programming languages, type disciplines, and reasoning methods.

"An in-place algorithm converts data structure using a minimal constant extra storage space. When such algorithms run, the input is overwritten by the output. For example, heap sort is an in-place sorting algorithm ".

" An operation is said to be an in-place operation if it does not alter the normal state of the system like a file backup can be stored over a running system without altering the speed of the system, while an in-place operation depends on the sophistication of the system".

In order to improve cache performance, an algorithm or application should increase data reuse, decrease cache conflicts, and decrease cache pollution, because a large amount of cache pollution increases the bandwidth requirement of the application, even though the application is not using more data [MPP02].

The technique, we use, is based on the principle of reusing, only, the available set of input variables in computing a mapping sequentially.

3.1.1 *In Situ* Design of Computation (IDC)

In this section, we explain the concept of *"In Situ Design of Computation (IDC)"* and interpret a single linear mapping.

Burckel et al. (see for instance [BGT09]) introduce a technique to compute mappings by sequence of linear assignments that are still linear mappings. This sequence of linear mappings reuses the available set of input variables and does not use any extra variable. This method of computation is known as *"In Situ Design of Computation (IDC)"* of mappings. An ultimate effect of this technique results in sequential break down of operations.

It starts in constructing the first assignment from matrix that represents coefficients of the given mapping, then compute its reference value that use in the next assignments where ever it needed to be referenced. Each linear assignment, involved in the sequence, is restricted to have maximum number of variables, not more than as available in the initial set of input variables. Each assignment is constructed in such a way that its reference can be computed and the computed reference is used in next assignments involved in the sequence, if it is required there.

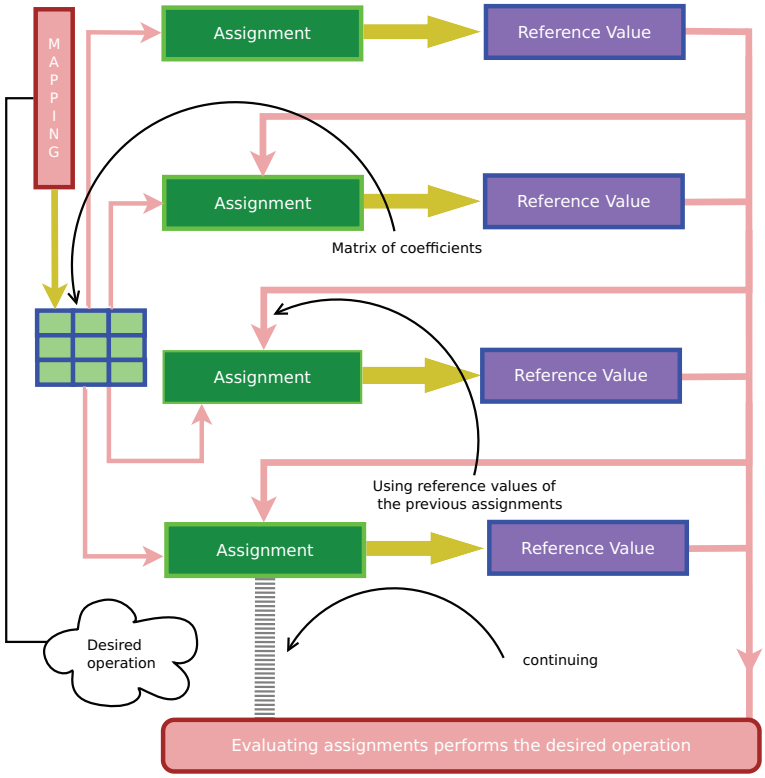


Figure 3.1: Sequential break-down of operations

An interpretation of the technique is shown in the Figure 3.1, where, assignments are shown by green bars and their corresponding reference values are presented with blue bars. When the references of previous assignments are used in the current assignments, it yields current transformation of mapping according to current values. In fact a single operation (computing mapping) is performed through sequence of assignments.

Interpreting a linear assignment

To interpret a linear assignment, a diagram is shown in Figure 3.2. Suppose that 'SOIV' denotes the set of input variables and 'v' denotes an *ith* variable of the set of input variables (SOIV). Delta contains elements of the set 'SOIV' but it does not depend on *ith* variable 'v', and 'alpha' (acts as coefficient for an assignment) denotes an element of field (say for instance) and it must be non-zero for an assignment to be invertible. A linear assignment consists in these parameters, is presented by green bar in the Figure 3.2. A small white gap in the dark brown block makes difference between 'SOIV' and 'Delta'.

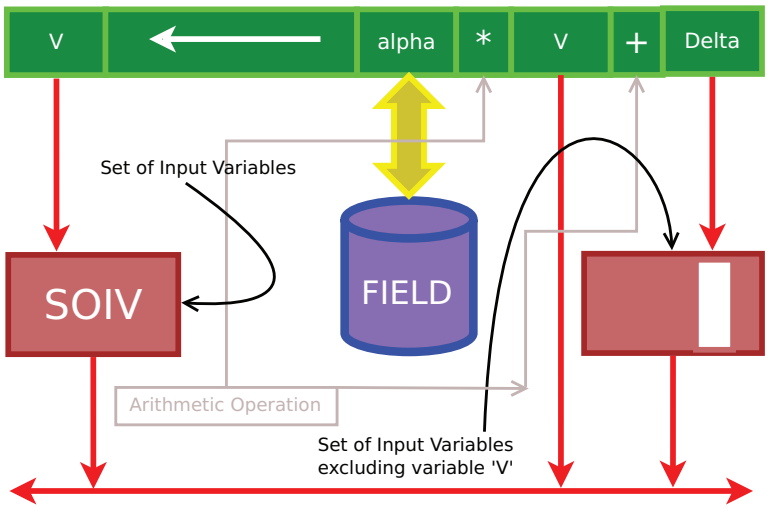


Figure 3.2: Interpreting linear assignment

How to compute reference value of a single assignment involving in sequence of assignments, is interpreted in Figure 3.3. In this Figure 3.3, 'beta' denotes the inverse value of 'alpha' that multiply with pink colored block of assignments, where, 'v' is the updated value of *ith* variable that could be used as reference value, if it is needed, in the next assignments involve in the sequence of assignments that compute a given mapping.

As an example, suppose that

$$\text{SOIV} = \{x_1,\, x_2,\, x_3, \ldots, x_{i-1},\, x_i,\, x_{i+1}, \ldots,\, x_n\}$$

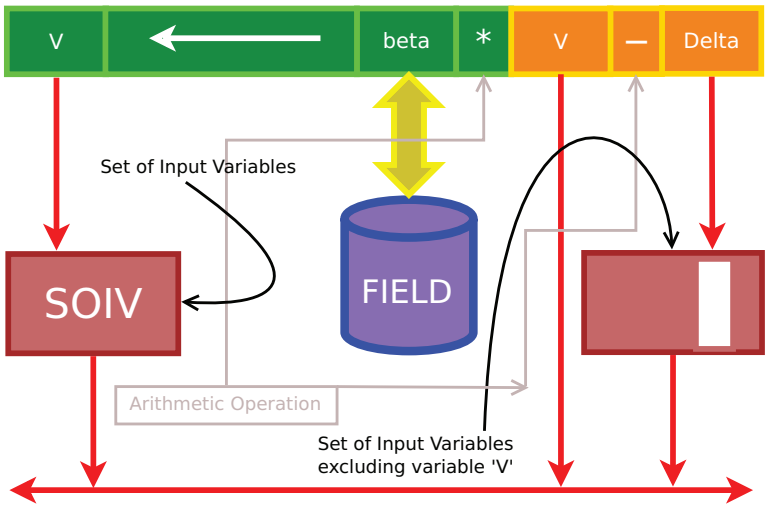


Figure 3.3: Reference value of linear assignment

For an *ith* variable $x_i \in$ SOIV, an assignment is

$$x_i := alpha(x_i) + Delta$$

where,

$$Delta = \{x_1, x_2, x_3, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n\}$$

The reference value is computed as

$$x_i := beta(x_i - Delta)$$

where,

$$beta = alpha^{-1} \neq 0$$

Next, we explain the existence of (IDC) for linear mappings over fields.

3.2 Existence of IDC for Linear Mappings over Fields

In this section, we explain that sequential computation of mappings by sequence of linear assignments (*In Situ* Design of Computation), such that the coefficients of these linear assignments belong to the field, exist. We explain the assertion (see [Bur07]) as follows.

Theorem 1. *Let E be a linear mapping on K^n, where K is a field and n is a positive integer. Then there exists a sequence $f_1, f_2, \ldots, f_{n-1}, f_n, g_{n-1}, \ldots, g_2, g_1$*

of linear mappings (assignments) from K^n to K such that the program :

$$\begin{aligned}
x_1 &:= f_1(x_1, x_2, \ldots, x_n)\\
x_2 &:= f_2(x_1, x_2, \ldots, x_n)\\
\vdots &:= \quad \vdots \quad \vdots \quad \vdots \quad \vdots\\
x_{n-1} &:= f_{n-1}(x_1, x_2, \ldots, x_n)\\
x_n &:= f_n(x_1, x_2, \ldots, x_n)\\
x_{n-1} &:= g_{n-1}(x_1, x_2, \ldots, x_n)\\
\vdots &:= \quad \vdots \quad \vdots \quad \vdots \quad \vdots\\
x_2 &:= g_2(x_1, x_2, \ldots, x_n)\\
x_1 &:= g_1(x_1, x_2, \ldots, x_n)
\end{aligned}$$

performs the operation $X := E(X)$ for any $X = (x_1, x_2, \ldots, x_n)$ in K^n .
Moreover:
For any i, g_i will be x_i or $x_i + x_j$ for some $j > i$.
E will be bijective $\Longleftrightarrow$ $f_i[i] \neq 0$ for any i.
For any i, if $f_i[i] = 0$ then $g_i = x_i$ and $f_j[i] = 0$ for any $j \geq i$.

Proof. For $a_{ji} \in K$, let $A = [a_{ji}]$, $(i, j = 1, 2, \ldots, n)$ be the matrix such that AX presents mapping E in a way that each row of matrix AX is the component of mapping E at current values of the variables $(x_1, x_2, \ldots, x_n)$. For example, the jth row $a_{j1}x_1 + a_{j2}x_2 + a_{j3}x_3 + \cdots + a_{jn}x_n$, of matrix AX will be the E_j component of mapping E. These linear mappings are denoted by F_i for $i = 1, 2, \ldots, n$ and the integers r_i are introduced to manage the second sequence. The first sequence f_i is computed by an iterative process as follows: For i from 1 to n, the following steps will be performed keeping $r_i = i$ at the beginning.

> Case 1: If $a_{ii} = \alpha \neq 0$, then $f_i := F_i$, and x_i will be modified to $x_i := \alpha x_i + \Delta$, where Δ does not depend on x_i. For any $a_{ji} \neq 0$, $j > i$, the reference $\alpha^{-1}(x_i - \Delta)$ is used to compute next mappings.
>
> Case 2: If $a_{ji} = \alpha = 0$, $\forall j \geq i$, then $f_i := F_i$, and $x_i := \Delta$.
>
> Case 3: If $a_{ii} = \alpha = 0$, and $a_{ji} \neq 0$, for some $j > i$, then select $a_{ji} = \beta$ such that $\beta \neq 0$. and perform operation $F_i := F_i - F_j$, *i.e.,* subtract jth row from ith row so that $\acute{\alpha} = -\beta \neq 0$, where $\acute{\alpha}$ denotes the new value of α. It leads to be again in case 1 and by the hypothesis the value $E_i - E_j$ will be assigned to x_i.
> The operation $F_i := F_i - F_j$ will introduce an assignment to the second sequence that can be obtained by adding E_j to x_i with $r_i := j$.

Computation of the second sequence:

The integers r_i, that are introduced, will be used to build the second sequence of assignments.
Therefore, from $i = n$ *to* 1, the following steps will be performed iteratively.

Step 1: If $r_i = i$, then x_i will be assigned in the first sequence and nothing to do more.

Step 2: If $r_i = j$ such that $j > i$. Then because it results by the operation $F_i := F_i - F_j$, therefore $g_i := x_i + x_j$, but for $i = n$, this situation will not exist.

The mapping E will not be injective in the case 2 of the computation of first sequence but if $a_{ii} \neq 0$ for any i, then one can compute E^{-1} by writing assignments, obtained, from last to first.
Compute $x_i := \alpha^{-1}(x_i - \Delta)$ from $x_i := \alpha x_i + \Delta$, $(\alpha \neq 0)$ and use these references in the next assignments to compute E^{-1}. □

Next, we explain Theorem 1 and the construction of sequence of linear assignments in detail.

3.2.1 Case-I

In this section, we illustrate, by the following example, how to construct sequence of linear assignments that involves in computing mapping sequentially.

Example 3. *Let E be a linear mapping defined as*

$$E\,(x_1,\, x_2,\, x_3) \longrightarrow (3x_1 + 7x_2 + 5x_3,\, 8x_1 + 4x_2 + 9x_3,\, 2x_1 + x_2 + 6x_3)$$

Then for $X := (x_1, x_2, x_3)$, the mapping E can be expressed as AX, where A denotes matrix of coefficients as given below.

$$A = \begin{bmatrix} 3 & 7 & 5 \\ 8 & 4 & 9 \\ 2 & 1 & 6 \end{bmatrix}$$

Now, since $\alpha = 3 \neq 0$, therefore, we are in the case-1. So we proceed in constructing the first assignment directly from the first row of the matrix A. It gives

$$x_1 := 3x_1 + 7x_2 + 5x_3 \tag{3.1}$$

To use the reference value of assignment 3.1 in the next computation, we compute the value for x_1 as given below.

$$\frac{1}{3}(x_1 - 7x_2 - 5x_3)$$

Now, for $\beta = 8$, we perform the operation

$$x_2 := x_2 - \beta x_1 + \beta\left\{\alpha^{-1}\,(x_1 - \Delta)\right\}$$

on second row of the matrix A and
for $\beta = 2$, we perform the operation

$$x_3 := x_3 - \beta x_1 + \beta\left\{\alpha^{-1}\,(x_1 - \Delta)\right\}$$

on third row of matrix A.
It gives,

$$x_2 := 8x_1 + 4x_2 + 9x_3 - 8x_1 + 8\left\{\frac{1}{3}(x_1 - 7x_2 - 5x_3)\right\}$$

$$x_2 := \frac{8}{3}x_1 - \frac{44}{3}x_2 - \frac{13}{3}x_3 \tag{3.2}$$

Similarly,

$$x_3 := 2x_1 + x_2 + 6x_3 - 2x_1 + \frac{2}{3}(x_1 - 7x_2 - 5x_3)$$

$$x_3 := \frac{2}{3}x_1 - \frac{11}{3}x_2 + \frac{8}{3}x_3$$

After performing these operations the matrix A takes the form

$$A = \begin{bmatrix} 3 & 7 & 5 \\ \frac{8}{3} & -\frac{44}{3} & -\frac{13}{3} \\ \frac{2}{3} & -\frac{11}{3} & \frac{8}{3} \end{bmatrix}$$

Now, the reference value x_2 for assignment 3.2, is given below,

$$\frac{8}{44}x_1 - \frac{3}{44}x_2 - \frac{13}{44}x_3$$

To compute third assignment, we proceed as follows.
For, $\alpha = -\frac{44}{3}$ and $\beta = -\frac{11}{3}$, perform the following operation:

$$x_3 := x_3 - \beta x_2 + \beta\left\{\alpha^{-1}(x_2 - \Delta)\right\}$$

$$x_3 := \frac{2}{3}x_1 - \frac{11}{3}x_2 + \frac{8}{3}x_3 + \frac{11}{3}x_2 - \frac{11}{3}\left\{\frac{8}{44}x_1 - \frac{3}{44}x_2 - \frac{13}{44}x_3)\right\}$$

It gives

$$x_3 := \frac{1}{4}x_2 + \frac{15}{4}x_3 \tag{3.3}$$

Combining (3.1), (3.2) and (3.3), we get the sequence of linear assignments, that compute the given mapping

$$\begin{aligned} x_1 &:= 3x_1 + 7x_2 + 5x_3 \\ x_2 &:= \frac{8}{3}x_1 - \frac{44}{3}x_2 - \frac{13}{3}x_3 \\ x_3 &:= \frac{1}{4}x_2 + \frac{15}{4}x_3 \end{aligned}$$

Computing Inverse mappings

Let S denotes the sequence of assignments that compute mapping E.

$$S = \begin{cases} x_1 := 3x_1 + 7x_2 + 5x_3 \\ x_2 := \frac{8}{3}x_1 - \frac{44}{3}x_2 - \frac{13}{3}x_3 \\ x_3 := \frac{1}{4}x_2 + \frac{15}{4}x_3 \end{cases}$$

To determine sequence of linear assignments that computes inverse mapping E^{-1}, we rewrite sequence S from bottom to top and invert each assignment $x_i := \alpha x_i + \Delta$ in the sequence S as $x_i := \alpha^{-1}\{x_i - \Delta\}$ such that we obtain new sequence $\acute{S}$ of assignments as given below.

$$\acute{S} = \begin{cases} x_3 := 0x_1 - \frac{1}{15}x_2 + \frac{4}{15}x_3 \\ x_2 := \frac{8}{44}x_1 - \frac{3}{44}x_2 - \frac{13}{44}x_3 \\ x_1 := \frac{1}{3}x_1 - \frac{7}{3}x_2 - \frac{5}{3}x_3 \end{cases}$$

By evaluating sequence $\acute{S}$ of linear assignments, we get the inverse mapping along with inverse of matrix A as given below.

$$A^{-1} = \begin{bmatrix} \frac{-1}{11} & \frac{37}{165} & -\frac{43}{165} \\ \frac{2}{11} & -\frac{8}{165} & -\frac{13}{165} \\ 0 & -\frac{1}{15} & \frac{4}{15} \end{bmatrix}$$

Further, it can be easily verified that $A * A^{-1} = I$ as given below.

$$A * A^{-1} = \begin{bmatrix} 3 & 7 & 5 \\ 8 & 4 & 9 \\ 2 & 1 & 6 \end{bmatrix} * \begin{bmatrix} \frac{-1}{11} & \frac{37}{165} & -\frac{43}{165} \\ \frac{2}{11} & -\frac{8}{165} & -\frac{13}{165} \\ 0 & -\frac{1}{15} & \frac{4}{15} \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = I$$

3.2.2 Case-II

In this section, we focus in explaining that, how to construct sequence of assignments when matrix of coefficients has not some non-zero entries at the position as described in Theorem 1 under case 2. We discuss this situation with the following example.

Example 4. *Consider a linear mapping*

$$E\,(x_1,\,x_2,\,x_3) \longrightarrow (5x_2 + 7x_3,\,x_2 + 4x_3,\,3x_1 + 2x_3)$$

Given mapping E can be written as $X := AX$ for a vector $X := (x_1, x_2, x_3)$ and a matrix A of coefficients, given as under

$$A = \begin{bmatrix} 0 & 5 & 7 \\ 0 & 1 & 4 \\ 3 & 0 & 2 \end{bmatrix}$$

Since $a_{11} = 0 = \alpha$. But $a_{13} = 3 = \beta$. Therefore we have to subtract third row of matrix A from first row. The operation will be $R_1 := R_1 - R_3$.
After the completion of this operation, the matrix A will take the form:

$$A = \begin{bmatrix} -3 & 5 & 5 \\ 0 & 1 & 4 \\ 3 & 0 & 2 \end{bmatrix}$$

Now the first assignment will be of the form:

$$x_1 := -3x_1 + 5x_2 + 5x_3 \tag{3.4}$$

The initial value of x_1 will be

$$\frac{1}{3}(-x_1 + 5x_2 + 5x_3)$$

Now, we will perform the operation

$$x_2 := x_2 - \beta x_1 + \beta\left\{\alpha^{-1}\,(x_1 - \Delta)\right\}$$

and

$$x_3 := x_3 - \beta x_1 + \beta\left\{\alpha^{-1}\,(x_1 - \Delta)\right\}$$

The matrix A will take the form

$$A = \begin{bmatrix} -3 & 5 & 5 \\ 0 & 1 & 4 \\ -1 & 5 & 7 \end{bmatrix}$$

The second assignment will be

$$x_2 := x_2 + 4x_3 \tag{3.5}$$

Now, the initial value of second assignment, x_2 will be

$$x_2 - 4x_3$$

and for the computation of third assignment:
For, $\alpha = 1$ and $\beta = 5$, We perform the following operation:

$$x_3 := x_3 - \beta x_2 + \beta \left\{ \alpha^{-1} (x_2 - \Delta) \right\}$$

We get the third assignment as

$$x_3 := -x_1 + 5x_2 - 13x_3 \tag{3.6}$$

The first assignment of second sequence that is obtained by the operation $R_1 := R_1 - R_3$ will be.

$$x_1 := x_1 + x_3$$

Hence the required sequence of assignments is

$$\textit{Sequence of assignments} \rightarrow \begin{cases} x_1 := -3x_1 + 5x_2 + 5x_3 \\ x_2 := x_2 + 4x_3 \\ x_3 := -x_1 + 5x_2 - 13x_3 \\ x_1 := x_1 + x_3 \end{cases}$$

3.2.3 Computing Inverse Mapping

The inverse mapping is computed by applying the same technique as described in section 3.2.1, i.e., by inverting the assignments and rewriting from bottom to top.

$$\textit{Sequence of assignments} \rightarrow \begin{cases} x_1 := x_1 - x_3 \\ x_3 := \dfrac{1}{13} (-x_1 + 5x_2 - x_3) \\ x_2 := \dfrac{4}{13} x_1 - \dfrac{7}{13} x_2 \\ x_1 := \dfrac{1}{3} (-x_1 + 5x_2 + 5x_3) \end{cases}$$

3.2.4 Case-III

Following example illustrates, how to construct sequence of assignments when the matrix of coefficients has not some non-zero entries at the position as described in Theorem 1 under case 3. As compared to the previous two cases, case 3 is easier.

Example 5. *Let E be a linear mapping defined as*

$$E(x_1, x_2, x_3) \longrightarrow (3x_1 + 5x_2 + 7x_3, x_2 + 4x_3, 2x_3)$$

The mapping E is expressed as $X := AX$, where $X := (x_1, x_2, x_3)$, is a vector and A is a matrix of coefficients as given below

$$A = \begin{bmatrix} 3 & 5 & 7 \\ 0 & 1 & 4 \\ 0 & 0 & 2 \end{bmatrix}$$

Since $a_{11} = 3 \neq 0 = \alpha$. Therefore the first assignment will be of the form:

$$x_1 := 3x_1 + 5x_2 + 7x_3 \tag{3.7}$$

The initial value of x_1 will be

$$\frac{1}{3}(x_1 - 5x_2 - 7x_3)$$

Since $\beta = 0$ both for second and third row, therefore, Matrix A will remain unchanged, and no reference of x_1 is used to compute the second assignment. The second assignment will be

$$x_2 := x_2 + 4x_3 \tag{3.8}$$

In computation of third assignment, we also do not need to use any reference of first or second assignment.

$$x_3 := 2x_3 \tag{3.9}$$

Therefore, for such type of mappings (as defined in the Example 5) the sequence of assignments can be written directly from the given mapping. Hence the required assignments are

$$\textit{Sequence of Assignments} \rightarrow \begin{cases} x_1 := 3x_1 + 5x_2 + 7x_3 \\ x_2 := x_2 + 4x_3 \\ x_3 := 2x_3 \end{cases}$$

Computing Inverse Mapping

Sequence of three assignments that computes the inverse mapping is given below.

$$\textit{Computing Inverse Mapping} \rightarrow \begin{cases} x_3 := \frac{1}{2}x_3 \\ x_2 := x_2 - 4x_3 \\ x_1 = \frac{1}{3}(x_1 - 5x_2 - 7x_3) \end{cases}$$

The inverse of matrix A is as follows.

$$A^{-1} = \begin{bmatrix} \frac{1}{3} & -\frac{5}{3} & \frac{13}{6} \\ 0 & 1 & -2 \\ 0 & 0 & \frac{1}{2} \end{bmatrix}$$

3.2.5 Possibility of Computing Inverse Mappings

There exist some of the mappings for which a sequence of linear assignments that computes corresponding inverse mapping does not exist. We prove by giving the following example that to compute inverse mapping, each assignment must be invertible.

Example 6. *Consider a linear mapping*

$$E\,(x_1,\, x_2,\, x_3) \longrightarrow (5x_2 + 7x_3,\, x_2 + 4x_3,\, 2x_3)$$

For a vector $X := (x_1, x_2, x_3)$*, the mapping* E *takes the form as* $X := AX$*, where* A *is a matrix of coefficients as given below.*

$$A = \begin{bmatrix} 0 & 5 & 7 \\ 0 & 1 & 4 \\ 0 & 0 & 2 \end{bmatrix}$$

Since $a_{11} = 0 = \alpha$ *and* $\beta = 0$*, for all other cases, therefore we do not need to perform any operation, and we construct the required assignments directly from matrix* A *as given below.*

$$\textit{Sequence of assignnments} \rightarrow \begin{cases} x_1 := 5x_2 + 7x_3 \\ x_2 := x_2 + 4x_3 \\ x_3 := 2x_3 \end{cases}$$

"Now, to compute inverse mapping, each assignment must be invertible".

Notice that one of the assignments (given above) is not invertible, and is given below.

$$x_1 := 5x_2 + 7x_3$$

Therefore, we cannot compute inverse mapping. Moreover, the matrix "A" is a singular matrix.

3.3 An approach using Bézout's Identity

In this section, we introduce an alternative approach (that use Bézout's Identity) for constructing a sequence of linear assignments that computes the given linear mapping. We prove it, here, for two dimensional linear mappings.

Definition 1. *Bézout's identity states that, if two integers* a *and* b *are relatively prime then there exists* $u, v \in \mathbb{Z}$ *such that* $au + bv = 1$*.*

Theorem 2. *Every linear mapping* E*:* $(x,\, y) \longrightarrow (mx + ny,\, px + qy)$*, where* $m, n, p, q \in \mathbb{Z}$*, can be computed by a sequence of at most* 3 *linear assignments with rational coefficients.*

Proof. Suppose that the given mapping E is computed with a sequence of three linear assignments as given below

$$\text{Three linear Assignnments} \rightarrow \begin{cases} x := ax + by \\ y := cx + dy \\ x := ex + fy \end{cases} \tag{S}$$

with rational coefficients a, b, c, d, e, and f. The sequence of linear assignments S (after evaluating sequentially) takes the form:

$$\begin{aligned} x &:= ax + by \\ y &:= cax + (cb + d)y \\ x &:= (ea + fca)x + (eb + fcb + fd)y \end{aligned}$$

Coefficients of given Linear mapping $E(x, y) \longrightarrow (mx + ny, px + qy)$, can be expressed as a matrix of order 2×2 as given below.

$$\begin{bmatrix} m & n \\ p & q \end{bmatrix} = A(say)$$

$$\text{Determinant (A)} = mq - np \tag{3.10}$$

We establish a system of four equations as given below:

$$\left.\begin{aligned} m &= ea + fca \\ n &= eb + fcb + fd \\ p &= ca \\ q &= bc + d \end{aligned}\right\} \tag{SOE}$$

By the equality 3.10 and the system of equations SOE, we have

$$mq - pn = aed$$

It yields that, the product of three variables a, e and d must be equal to determinant of matrix A. Now, the general solution of the system of equations SOE is as given below.

$$a = -\frac{fp - m}{e},\ b = -\frac{fq - n}{e},\ c = -\frac{pe}{fp - m},\ d = -\frac{mq - np}{fp - m} \tag{3.11}$$

To find particular solution, we modify matrix A using Bézout's identity. Suppose that m and p are relatively prime, then we find $u, v \in \mathbb{Z}$ such that $um + vp = 1$ and modify matrix A by multiplying its rows by u and v respectively. New matrix will satisfy conditions 3.11 and yields values for coefficients. To return back to the original matrix, we perform the following operations.

$$y := y(\frac{1}{v}) \quad \text{and} \quad x := x(\frac{1}{u})$$

If m and p are not co-prime, then we make them co-prime by extracting their GCD and solve the mapping as, for co-prime case and finally assign the GCD to the coefficient a. □

Next, we explain, Theorem 2 in detail, by examples.

3.3.1 Case-I

In this section, we discuss the case when the GCD of entries of first column of the matrix (that represents coefficients of mapping), equals 1.

Example 7. *Let E be a linear mapping as defined below*

$$E : (x, y) \longrightarrow (55x + 89y, 34x + 21y)$$

such that matrix A represents the coefficients.

$$A = \begin{bmatrix} m & n \\ p & q \end{bmatrix} = \begin{bmatrix} 55 & 89 \\ 34 & 21 \end{bmatrix}$$

The values corresponding to the entries m and p are co-prime, i.e., $GCD(55, 34) = 1$. Therefore, we apply Bézout's identity to find integers u and v such that $55u + 89v = 1$
A number of values are possible for u and v. Two of them are $u = 13$, $v = -21$.
Multiply first row of matrix A by u and second by v, it gives

$$\acute{A} = \begin{bmatrix} um & un \\ vp & vq \end{bmatrix} = \begin{bmatrix} 715 & 1157 \\ -714 & -441 \end{bmatrix}$$

Using

$$a = -\frac{fp - m}{e},\ b = -\frac{fq - n}{e},\ c = -\frac{pe}{fp - m},\ d = -\frac{mq - np}{fp - m} \tag{3.12}$$

we find the values $a = 1$, $b = 716$, $c = -714$, $d = 510783$, $e = 1$, $f = -1$
Therefore, the sequence of assignments is as given below:

$$\begin{cases} x := x + 716y \\ y := \dfrac{1}{13}\left(-714x + 510783y\right) \\ x := -\dfrac{1}{21}\left(x - y\right) \end{cases}$$

Next, we give another example to explain the case, when the entries of the first column of matrix A are not co-prime

3.3.2 Case-II

In this section, we discuss the case when the GCD of entries of the first column of matrix (that represent coefficients of mapping), is not equals 1.

Example 8. *Consider the mapping E:* $(x, y) \longrightarrow (25x + 13y, 35x + 21y)$, *whose coefficients are expressed by matrix A as given below.*

$$A = \begin{bmatrix} m & n \\ p & q \end{bmatrix} = \begin{bmatrix} 25 & 13 \\ 35 & 21 \end{bmatrix}$$

Notice that $GCD(m, p) = GCD(25, 35) = 5 \neq 1$. *We extract GCD to make entries of the column co-prime*

$$B = \begin{bmatrix} m & n \\ p & q \end{bmatrix} = \begin{bmatrix} 5 & 13 \\ 7 & 21 \end{bmatrix}$$

Now $GCD(5, 7) = 1$, *so solving Bézout's identity* $5u + 7v = 1$, *one of the possible solutions is* $u = -4$, $v = 3$. *Multiply first row of the matrix B with u and second row with v, It gives*

$$B = \begin{bmatrix} um & un \\ vp & vq \end{bmatrix} = \begin{bmatrix} -20 & -52 \\ 21 & 63 \end{bmatrix}$$

For matrix B, the system of equations SOE takes the form

$$\left.\begin{array}{ll} -20 & = ea + fca \\ -52 & = eb + fcb + fd \\ 21 & = ca \\ 63 & = bc + d \end{array}\right\} \qquad \text{(SOE-1)}$$

Solving the system of equations SOE-1, we find the values

$$a = 1,\ b = 11,\ c = 21$$
$$d = -168,\ e = 1,\ f = -1$$

The sequence of assignments for $a = 1$, *is given below.*

$$\begin{aligned} x &:= x + 11y \\ y &:= -\frac{1}{4}(21x - 168y) \\ x &:= \frac{1}{3}(x - y) \end{aligned}$$

Now, replace $a = 5$, *and the required sequence of assignments for* $a = 5$, *is as under*

$$\begin{cases} x := 5x + 11y \\ y := -\dfrac{1}{4}(21x - 168y) \\ x := \dfrac{1}{3}(x - y) \end{cases}$$

3.3.3 Computing Inverse Mappings:

In this section, we show by example, how to compute inverse mappings.

Example 9. *The intermediate sequence of assignments that compute mapping E, in Example 7, whose coefficients are represented by matrix*

$$\acute{A} = \begin{bmatrix} 715 & 1157 \\ -714 & -441 \end{bmatrix}$$

is given as under

$$\begin{cases} x := x + 716y \\ y := -714x + 510783y \\ x := x - y \end{cases} \tag{SOA}$$

We construct the sequence of assignments that compute inverse mapping, by inverting and rewriting the sequence of assignments SOA, as given below.

$$\begin{aligned} x &:= x + y \\ y &:= \frac{1}{510783}\{714x + y\} \\ x &:= x - 716y \end{aligned}$$

and the inverse matrix $\acute{A}^{-1}$ is obtained by evaluating the sequence of assignments that compute inverse mapping.

$$\acute{A}^{-1} = \begin{bmatrix} -\frac{21}{24323} & -\frac{89}{39291} \\ \frac{34}{24323} & \frac{55}{39291} \end{bmatrix}$$

We compute A^{-1} by making some changes in the above sequence of assignments, i.e., we multiply the first column of $\acute{A}^{-1}$ by u and second by v. The sequence of assignments that compute A^{-1} is given below:

$$\begin{aligned} x &:= u * x + v * y \\ y &:= \frac{1}{510783}\{714x + v * y\} \\ x &:= x - 716y \end{aligned}$$

The matrix A^{-1} corresponding to matrix A is given below:

$$A^{-1} = \begin{bmatrix} -\frac{21}{1871} & \frac{89}{1871} \\ \frac{34}{1871} & -\frac{55}{1871} \end{bmatrix}$$

Chapter 4

In Situ Design of Computation for Linear Mappings over Rings

Contents

4.1 Existence of IDC for Linear Mappings over Rings 48

4.1.1 Assignment Matrices 48

4.2 Explanation and Construction 50

4.2.1 Case-I 50

4.2.2 Case-II 52

4.3 Computing Inverse Mappings 54

4.3.1 Possibility of computing Inverse mappings . . . 55

Investigating the existence of *"In Situ Design of Computation (IDC)"* for linear mappings over rings consists, actually, in finding whether the coefficients of such *"In Situ Design of Computation (IDC)"* belong to rings or not. The existence of IDC of linear mappings over rings is proved in [BGT09]. We are interested to verify and implement the existence of *"In Situ Design of Computation (IDC)"* over rings.

In this chapter, we explain the existence of *"In Situ Design of Computation (IDC)"* for linear mappings over rings. We design algorithm (given in annex B) to verify and implement the idea. Section 4.1 consists in explaining the existence of such type of computations over rings, in detail. How a matrix is decomposed into assignment matrices and how the corresponding assignments can be constructed, is described in section 4.2. We discuss the possibility of constructing sequence of assignments that compute inverse mappings in section 4.3 and provide counter example.

4.1 Existence of IDC for Linear Mappings over Rings

It is proved (see [BGT09]) that *"In Situ Design of Computation (IDC)"* for linear mappings such that the coefficients of these linear mappings belong to rings, exist. In this section we explain the idea for the ring $\mathbb{Z}/N\mathbb{Z}$.

Lemma 1 ([BGT09]). *Suppose that $x_1, \cdots, x_n$ be co-prime integers and N be an integer. Then, there exists integers $\lambda_2, \cdots, \lambda_n$ such that $x_1 + \Sigma_i \lambda_i x_i \in (\mathbb{Z}/N\mathbb{Z})^*$, where $(\mathbb{Z}/N\mathbb{Z})^*$ denotes the group of invertible elements.*

Proof. Suppose that N be a prime power p^v. It is given that integers x_i are co-prime, therefore there exists an integer i_0 (say) such that x_{i0} is co-prime to p. If x_1 is itself co-prime to p, then one can select $i_0 = 1$. If x_1 is not co-prime to p, *i.e.*, if x_1 is divisible by p, then $x_1 + x_{i0}$ is co-prime to p. So the result holds when $N = p^v$. For each prime power dividing N , we can therefore construct n integers $\lambda_1^{p^v}, \cdots, \lambda_n^{p^v}$ such that

$$\lambda_1^{p^v} = 1 \text{ and } \Sigma_i (\lambda_i)^{p^v} x_i \in (\mathbb{Z}/p^v\mathbb{Z})^*$$

Applying Chinese Remainder Theorem, these vectors combine to form a global solution

$$(1, \lambda_2, \cdots, \lambda_N)$$

satisfying the required property. □

4.1.1 Assignment Matrices

In this section, we explain the basic idea of assignment matrices and the possibility of decomposing a matrix into assignment matrices under "modulo" operation. An assignment matrix is actually the modified form of identity matrix with a row different from identity matrix as defined below.

Definition 2. *A matrix A is said to be an assignment matrix, if there exists an integer i_0 such that for all row and column indices (i, j), one has either $i = i_0$ or $A_{i,j} = \delta_i^j$.*

By definition 2, for a square matrix A, $A - I$ has at most one non-zero row, where I denotes the identity matrix.

Example 10. *Let A be the square matrix as given below:*

$$A = \begin{bmatrix} 2 & 3 & 5 \\ 4 & 7 & 1 \\ 5 & 6 & 7 \end{bmatrix}$$

Performing operation with "modulo 8", the given matrix A is decomposed into four assignment matrices as given below.

$$A_1 := \begin{bmatrix} 7 & 1 & 4 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, A_2 := \begin{bmatrix} 1 & 0 & 0 \\ 4 & 3 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

$$A_3 := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 7 & 1 & 2 \end{bmatrix}, A_4 := \begin{bmatrix} 1 & 0 & 7 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

such that

$$A_4 * A_3 * A_2 * A_1 = A \ (modulo\ 8)$$

Next, we explain the assertion regarding existence of *"In Situ Design of Computation (IDC)"* over rings.

Proposition 1 (see [BGT09])**.** *Let N be an integer. Any $n \times n$ matrix over $(\mathbb{Z}/N\mathbb{Z})$ can be written as the product of at most $2n-1$ assignment matrices.*

Proof. The result can be easily proved by the help of induction on the number of rows (representing by n) of the given matrix. Therefore if $n = 1$, the result is obvious.
For $n > 1$, we can proceed as follows. Consider the first column of the given matrix "A" (say). Suppose that the entry a_{11} of the matrix "A" is not an invertible element under modulo N. Let 'g' denotes GCD of the first column, *i.e.*, $g = (a_{11}, a_{21}, \cdots, a_{n1})$. We construct an invertible element using Lemma 1 and we apply it to make a combination of the coefficients of this column equal to g times an invertible element of $(\mathbb{Z}/N\mathbb{Z})$, with the constraint that this combination has its first multiplier equal to 1. This implies that the $n \times n$ matrix T defined by $t_{i,j} = \delta_i^j$ for $i > 1$, and $t_{1,j} = \lambda_j$, where the multipliers $1, \lambda_2, \cdots, \lambda_n$ are obtained from Lemma 1. Clearly, matrix T is an invertible assignment matrix, and the product $T * A$ has a coefficient at position (1, 1) which is equal to g times an invertible element modulo N. Now assuming that $a_{1,1} \in g(\mathbb{Z}/N\mathbb{Z})^*$. Let n' be the number of columns of A . Let $G = diag(g, 1, \cdots, 1)$, and let A' be the integer matrix AG^{-1} (A' has coefficients in $\mathbb{Z}$ because g is the GCD of the first column). We have $a'_{1,1} \in (\mathbb{Z}/N\mathbb{Z}^*)$. We form an $n' \times n'$ assignment matrix U defined by $u_{i,j} = \delta_i^j$ for $i > 1$, and $u_{1,j} = a'_{1,j}$. The matrix U is invertible modulo N (its determinant is $a'_{1,1}$). The first row of the matrix $A'' = A' * U^{-1}$ is equal to $(1, 0, \cdots, 0)$. Notice further that UG is an assignment matrix as well (even though not invertible modulo N). Putting together the different results we have that $A = T \times A'' \times (UG)$, where the matrix T may be omitted. Applying the result inductively on A'' completes the proof. □

Next, we explain Proposition 1 in detail and describe method for constructing the sequence of linear assignments.

4.2 Explanation and Construction

This section consists in explaining, how to implement Proposition 1 in constructing sequence of assignments.

Let "A" be the matrix that represents coefficients of the given mapping. Now, there are two cases, whether the entry "$a_{1,1}$" of matrix "A" is invertible under chosen value of modulo or not.

If the entry "$a_{1,1}$" of the matrix "A" is invertible under chosen value of the modulo then matrix T will be the identity matrix of the same order as of matrix A and $A' = AG^{-1}$.

Otherwise, we construct matrix T as defined below

$$t_{ij} = \delta_{ij} \text{ for } i > 1 \text{ and } t_{1,j} = \lambda_j$$

By definition, first row of the matrix T will consist of the multipliers $1, \lambda_1, \cdots, \lambda_n$, obtained by Lemma 1. It is noted that matrix T could be single matrix or $T = T_1 \cdot T_2, \cdots, T_n$ depending on number of prime factors of N.
An invertible matrix L can be constructed by solving the system $TL = A$.
i.e.

$$L_1 = T_1 A,\ L_2 = T_2 A, \cdots, L_n = T_n A$$

and

$$A = (T_1 * T_2, \cdots, T_n) L_n$$

where "g" denotes GCD of the first column of matrix L_i obtained after T_i transformation. So if

$$A = (T_1 * T_2, \cdots, T_n) L_f$$

then $A' = L_f G^{-1}$, where L_f denotes the final invertible matrix after T_i, $1 \leq i \leq n$ transformation.

Matrix A will finally satisfy the relation $A = (T_1 * T_2, \cdots, T_n) A'' UG$.

Next, we show by examples, how these assignment matrices can be constructed.

4.2.1 Case-I

In this section, we explain the construction of sequence of linear assignments, when N is the product of single integer.

Example 11. *Consider a mapping* $E : (x_1, x_2) \longrightarrow (2x_1 + 3x_2, 5x_1 + 7x_2)$. *The coefficients can be represented by a square matrix*

$$M = \begin{bmatrix} 2 & 3 \\ 5 & 7 \end{bmatrix}$$

Suppose that $N = 8 = 2^3$.
For $p^v = 2$, $(2,2) \neq 1, \Longrightarrow m_{1,1}$ *is not invertible.*
Since $\lambda_2 = 1$, *therefore matrix*

$$T = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

Now, solving the system as given below

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} 2 & 3 \\ 5 & 7 \end{bmatrix}$$

We have values for coefficients a, b, c, *and* d *as given below.*

$$a = -3 = 5 \ (mod\ 8)$$

$$b = -4 = 4 \ (mod\ 8)$$

$$c = 5,\ d = 7$$

Now we construct matrix M *as*

$$M = \begin{bmatrix} 5 & 4 \\ 5 & 7 \end{bmatrix}$$

Further,

$$G = \begin{bmatrix} 5 & 0 \\ 0 & 1 \end{bmatrix}$$

where,

$$G^{-1} = 5^{-1} \begin{bmatrix} 1 & 0 \\ 0 & 5 \end{bmatrix} = \begin{bmatrix} 5 & 0 \\ 0 & 1 \end{bmatrix} (mod\ 8)$$

$$\begin{aligned} M' = MG^{-1} &= \begin{bmatrix} 5 & 4 \\ 5 & 7 \end{bmatrix} \begin{bmatrix} 5 & 0 \\ 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 25 & 4 \\ 25 & 7 \end{bmatrix} = \begin{bmatrix} 1 & 4 \\ 1 & 7 \end{bmatrix} (mod\ 8) \\ U &= \begin{bmatrix} 1 & 4 \\ 0 & 1 \end{bmatrix} \\ U^{-1} &= \begin{bmatrix} 1 & -4 \\ 0 & 1 \end{bmatrix} \\ UG &= \begin{bmatrix} 1 & 4 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 5 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 5 & 4 \\ 0 & 1 \end{bmatrix} \\ M'' = M'U^{-1} &= \begin{bmatrix} 1 & 4 \\ 1 & 7 \end{bmatrix} \begin{bmatrix} 1 & -4 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & 3 \end{bmatrix} \end{aligned}$$

Notice that matrix M is decomposed into three assignment matrices T, M'' and UG such that

$$TM''UG = \begin{bmatrix} 10 & 11 \\ 5 & 7 \end{bmatrix} = \begin{bmatrix} 2 & 3 \\ 5 & 7 \end{bmatrix} (mod\ 8)$$

These three assignment matrices yield the sequence of linear assignments that compute the given mapping E and is given below

$$Sequence\ of\ assignments \rightarrow \begin{cases} a := 5a + 4b \\ b := a + 3b \\ a := a + b \end{cases}$$

4.2.2 Case-II

In this section, we explain the construction of sequence of linear assignments, when N is not the product of single integer.

Example 12. *Consider a mapping $E : (x_1, x_2) \longrightarrow (2x_1 + 3x_2, 5x_1 + x_2)$. The coefficients can be represented by a square matrix*

$$M = \begin{bmatrix} 2 & 3 \\ 5 & 1 \end{bmatrix} and \quad N = 6 = 2^1 * 3^1$$

$m_{1,1}$ is not invertible with respect to $p^v = 2$, $(2, 2) \neq 1$
$\lambda_1 = 1$, and $\lambda_2 = 1$,

$$T_1 = \begin{bmatrix} \lambda_1 & \lambda_2 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

Now, by solving the following system

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} 2 & 3 \\ 5 & 1 \end{bmatrix}$$

We have values for coefficients a, b, c, and d as given below.

$$a = -3 = 3\ (mod\ 6)$$

$$b = 2,\ c = 5,\ and\ d = 1$$

Then we proceed by constructing matrix

$$L_1 = \begin{bmatrix} 3 & 2 \\ 5 & 1 \end{bmatrix}$$

Observe that entry $l_{1,1}$ is not invertible, For $p^v = 3$, $(3, 3) \neq 1$
$\lambda_1 = 1$, and $\lambda_2 = 1$,

$$T_2 = \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix}$$

Again, solving the system

$$\begin{bmatrix}1 & 2\\0 & 1\end{bmatrix}\begin{bmatrix}a & b\\c & d\end{bmatrix}=\begin{bmatrix}3 & 2\\5 & 1\end{bmatrix}$$

We have values for coefficients a, b, c, and d as given below.

$$a=-7=5\ (mod\ 6)$$

$$b=0,\ c=5,\ and\ d=1$$

Now matrix

$$L_2=\begin{bmatrix}5 & 0\\5 & 1\end{bmatrix}$$

$$G=\begin{bmatrix}5 & 0\\0 & 1\end{bmatrix}$$

where,

$$\begin{aligned}G^{-1}&=5^{-1}\begin{bmatrix}1 & 0\\0 & 5\end{bmatrix}=5\begin{bmatrix}1 & 0\\0 & 5\end{bmatrix}\\&=\begin{bmatrix}5 & 0\\0 & 1\end{bmatrix}\ (mod\ 6)\\M'=MG^{-1}&=\begin{bmatrix}5 & 0\\5 & 1\end{bmatrix}\begin{bmatrix}5 & 0\\0 & 1\end{bmatrix}\\&=\begin{bmatrix}25 & 0\\25 & 1\end{bmatrix}=\begin{bmatrix}1 & 0\\1 & 1\end{bmatrix}\ (mod\ 6)\end{aligned}$$

Now,

$$\begin{aligned}U&=\begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}\\U^{-1}&=\begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}\\UG&=\begin{bmatrix}5 & 0\\0 & 1\end{bmatrix}\\M''=M'U^{-1}&=\begin{bmatrix}1 & 0\\1 & 1\end{bmatrix}\end{aligned}$$

Thus, the matrix M is decomposed into three assignment matrices $T=T_1*T_2$, M'' and UG such that

$$T_1T_2M''UG=\begin{bmatrix}20 & 3\\5 & 1\end{bmatrix}=\begin{bmatrix}2 & 3\\5 & 1\end{bmatrix}\ (mod\ 6)$$

From these assignment matrices, the sequence of linear assignments (that computes the given mapping) is constructed and is given below:

$$Sequence\ of\ assignments\rightarrow\begin{cases}a:=5a\\b:=a+b\\a:=a+3b\end{cases}$$

4.3 Computing Inverse Mappings

In this section, we show, by examples, how to construct a sequence of linear assignments that computes inverse mapping, by a sequence of linear assignments over the ring $\mathbb{Z}/N\mathbb{Z}$.

Example 13. *Let E be a linear mapping defined as follows*

$$E\,(x_1,\, x_2,\, x_3) \longrightarrow (2x_1 + 8x_2 + 6x_3,\, 3x_1 + 13x_2 + 7x_3,\, 5x_1 + 5x_2 + x_3)$$

and "A" be the matrix expressing coefficients of mapping E, given as under

$$A = \begin{bmatrix} 2 & 8 & 6 \\ 3 & 13 & 7 \\ 5 & 5 & 1 \end{bmatrix}$$

Suppose that we are to perform operations under "modulo 9". Then

$$A = \begin{bmatrix} 2 & 8 & 6 \\ 3 & 4 & 7 \\ 5 & 5 & 1 \end{bmatrix} \; (\textit{modulo 9})$$

We construct the sequence of linear assignments that compute mapping E, under "modulo 9" operation, and is given below

$$\left.\begin{array}{l} x_1 := 2x_1 + 8x_2 + 6x_3 \\ x_2 := 6x_1 + x_2 + 7x_3 \\ x_3 := 7x_1 + 3x_2 + x_3 \end{array}\right\} \; (\textit{modulo 9}) \qquad \text{(S-1)}$$

We are interested in computing a sequence of linear assignments under "modulo 9" that compute inverse mapping E^{-1}. We obtain this sequence of assignments by inverting and rewriting each assignment of the Sequence S-1 (from bottom to top) that computes the given mapping E.

$$\textit{Sequence of assignments} \rightarrow \left\{\begin{array}{l} x_3 := x_3 - 7x_1 - 3x_2 \\ x_2 := x_2 - 6x_1 - 7x_3 \\ x_1 := 2^{-1}\,(x_1 - 8x_2 - 6x_3) \end{array}\right. \qquad \text{(S'-1)}$$

Replacing "2^{-1}" in the last assignment of the Sequence S'-1 of assignments by its inverse "modulo 9", we obtain the required sequence of assignments.

$$\left.\begin{array}{l} x_3 := x_3 - 7x_1 - 3x_2 \\ x_2 := x_2 - 6x_1 - 7x_3 \\ x_1 := 5\,(x_1 - 8x_2 - 6x_3) \end{array}\right\} \textit{modulo 9}$$

that computes inverse mapping. By, evaluating above sequence of linear assignments, we get the inverse of matrix A as given below.

$$A^{-1} = \begin{bmatrix} 7 & 2 & 7 \\ 7 & 4 & 2 \\ 2 & 6 & 1 \end{bmatrix}$$

Verifying the product of assignment matrices

The matrix

$$A = \begin{bmatrix} 2 & 8 & 6 \\ 3 & 13 & 7 \\ 5 & 5 & 1 \end{bmatrix}$$

is decomposed into three assignment matrices under modulo 9 *operation, these matrices are as given below:*

$$A_1 = \begin{bmatrix} 2 & 8 & 6 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} A_2 = \begin{bmatrix} 1 & 0 & 0 \\ 3 & 13 & 7 \\ 0 & 0 & 1 \end{bmatrix} A_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 5 & 5 & 1 \end{bmatrix}$$

The product of matrices is $A_3 * A_2 * A_1 = A$.

4.3.1 Possibility of computing Inverse mappings

In this section, we provide example which proves that constructing a sequence of linear assignments that computes the inverse mapping of a given mapping may not exist.

Definition 3. *A modular inverse of an integer "a" (modulo k) is the integer "a^{-1}" such that*

$$aa^{-1} \equiv 1 \; mod \; k$$

i.e, an integer "a" has an inverse (modulo k) if and only if $GCD(a, k) = 1$. *If "p" is a prime number, then for each* $a \not\equiv 0$ *(modulo p), there exists a multiplicative inverse mod p.*

It is not always true that an integer "a" with $a \not\equiv 0 \pmod{k}$ has an inverse mod k.

Consider, for instance, $2 \not\equiv 0 \pmod 4$ does not have modular inverse because of

$$2 \times 2 = 4 \equiv 0 \bmod 4$$

Thus (2 mod 4) has no modular inverse, otherwise, we could multiply both sides of

$$2 \times 2 \equiv 0 \; mod \; 4$$

by the inverse of 2 and get the false result $2 \equiv 0 \bmod 4$.

Next, we give an example to show that there may not exist a sequence of linear assignments that required to compute inverse mapping.

Example 14. *Let E be the linear mapping defined as under.*

$$E\,(x_1,\, x_2,\, x_3) \longrightarrow (2x_1 + 8x_2 + 6x_3,\, 3x_1 + 13x_2 + 7x_3,\, 5x_1 + 5x_2 + x_3)$$

The mapping E has already discussed in Example 13 with "modulo 9". Suppose that we want to construct the sequence of linear assignments keeping operation under "modulo 12". Then the required sequence of assignments is given below

$$\textit{Sequence of assignments} \rightarrow \begin{cases} x_1 := 5x_1 + 5x_2 + 9x_3 \\ x_2 := 3x_1 + 10x_2 + 4x_3 \\ x_3 := x_1 + 4x_3 \\ x_1 := x_1 + 3x_2 \end{cases}$$

To construct sequence of assignments that computes inverse mapping E^{-1}, we need to invert each assignment and to rewrite the above sequence of assignments from bottom to top. Performing this technique, we get sequence of assignments as given below.

$$\textit{Sequence of assignments} \rightarrow \begin{cases} x_1 := x_1 - 3x_2 \\ x_3 := 4^{-1}\,(x_3 - x_1) \\ x_2 := 10^{-1}\,(x_2 - 3x_1 - 4x_3) \\ x_1 := 5^{-1}\,(x_1 - 5x_2 - 9x_3) \end{cases}$$

Observe that, the second and third assignment of the above sequence is not invertible due to the reason that integers 4 *and* 10 *have not their modular inverses under "modulo 12" operation.*

Similarly, the inverse of the matrix (that presents coefficients of the mapping E) does not have its inverse under "modulo 12" because the determinant of matrix A

$$|A| = \begin{vmatrix} 2 & 8 & 6 \\ 3 & 13 & 7 \\ 5 & 5 & 1 \end{vmatrix} = 248$$

is not invertible under "modulo 12" operation.

"Therefore, to compute inverse mapping for a mapping, each linear assignment in the sequence of linear assignments (that compute the mapping) must be invertible".

Chapter 5

In Situ Design of Computation for Linear Mappings over Integers

Contents

5.1 Existence of IDC for Linear Mappings over Integers 58

5.1.1 Modifications: 60

5.1.2 Explanation and construction of assignments: . 60

5.2 Investigating bounds for the number of assignments: 68

5.2.1 An approach based on Fibonacci numbers 71

5.2.2 Identity (coefficient linkage): 74

Proving the existence of *"In Situ Design of Computation (IDC)"* over the set of integers (to compute mappings) leads to perform these operations in more simple way as compared to IDC for mappings over fields and rings. In this chapter, (following the ideas of Serge Burckel), we prove the existence of *"In Situ Design of Computation (IDC)"* over the set of integers (to compute mappings sequentially). Section 5.1 describes the existence of *"In Situ Design of Computation (IDC)"* for linear mappings over the set of integers and explains the construction of assignments involve in IDC. Section 5.2 consists in investigating bound over the number of assignments required to compute mappings by IDC. An approach with Fibonacci numbers is attempted in the same section. At the end of this section, we prove an identity that relates the determinant of matrices (presenting coefficients of mappings) to the product of coefficients of assignments involve in computing the given mappings.

5.1 Existence of IDC for Linear Mappings over Integers

In this section, following the ideas of Serge Burckel, we provide the existence of *"In Situ Design of Computation (IDC)"* for linear mappings over integers. The assertion is proved as the following theorem.

Theorem 3. *Let E be a linear mapping on $\mathbb{Z}^n$. There exists a finite sequence of linear assignments of the form:*

$$\begin{aligned}
x_{p_1} &:= x_{p_1} + f_{p_1}(x_1, \ldots, x_{p_1-1}, x_{p_1+1}, \ldots, x_n)\\
x_{p_2} &:= x_{p_2} + f_{p_2}(x_1, \ldots, x_{p_2-1}, x_{p_2+1}, \ldots, x_n)\\
\vdots &:= \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots\\
x_{p_m} &:= x_{p_m} + f_{p_m}(x_1, \ldots, x_{p_m-1}, x_{p_m+1}, \ldots, x_n)\\
x_n &:= g_n(x_1, x_2, \ldots, x_n)\\
\vdots &:= \quad \vdots \quad \vdots \quad \vdots \quad \vdots\\
x_2 &:= g_2(x_1, x_2, \ldots, x_n)\\
x_1 &:= g_1(x_1, x_2, \ldots, x_n)
\end{aligned}$$

that computes the mapping E, where $p_1, \ldots, p_m \in \{1, 2, \ldots, n\}$ m, $n \in \mathbb{Z}$.

Proof. The given mapping E can be expressed as AX, where A is a square matrix of integer coefficients (*i.e.,* for $A = [a_{ij}]$, $a_{ij} \in \mathbb{Z}$, $i, j = 1, 2, \ldots, n$), such that each row of the matrix AX represents a component of the mapping E at current values of the variables $(x_1, x_2, \ldots, x_n)$ and is a linear mapping. We construct the sequence of linear assignments by considering the following four different cases:

Case 1: If $a_{ij} < 0$, for $i \geq j$, then multiply jth column by -1, so that $a_{ij} > 0$ and construct the corresponding linear assignment x_j as $x_j := -x_j$.

Case 2: If $a_{ii} > a_{ij}$, $for\, j > i$, then perform the operation $C_i := C_i - C_j$, and the corresponding linear assignment x_j will be of the form $x_j := x_j + f$, where f does not depend on x_j and C_i is the ith column of matrix A.

Case 3: If $a_{ii} \leq a_{ij}$, $for\, j > i$, then perform the operation $C_j := C_j - C_i$, and corresponding linear assignment x_i will take the form $x_i := x_i + f$.

Case 4: If $a_{ii} = 0$, then we perform the operation $C_i := C_i - C_j$, and x_j will take the form $x_j := x_j + f$.

Performing this procedure iteratively, we transform the square matrix A into triangular matrix L (say) as given below.

$$L := \begin{bmatrix} l_{11} & 0 & 0 & \dots & 0 & 0 & 0 \\ l_{21} & l_{22} & 0 & \dots & 0 & 0 & 0 \\ l_{31} & l_{32} & l_{33} & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ l_{n-2,1} & l_{n-2,2} & l_{n-2,3} & \dots & l_{n-2,n-2} & 0 & 0 \\ l_{n-1,1} & l_{n-1,2} & l_{n-1,3} & \dots & l_{n-1,n-2} & l_{n-1,n-1} & 0 \\ l_{n1} & l_{n2} & l_{n3} & \dots & l_{n,n-2} & l_{n,n-1} & l_{nn} \end{bmatrix}$$

We construct the second sequence of linear assignments directly from matrix L as follows:

$$\begin{aligned} x_n &:= l_{n1}x_1 + l_{n2}x_2 + l_{n3}x_3 + \ldots + l_{n,n-2}x_{n-2} + l_{n,n-1}x_{n-1} + l_{nn}x_n \\ x_{n-1} &:= l_{n-1,1}x_1 + l_{n-1,2}x_2 + \ldots + l_{n-1,n-2}x_{n-2} + l_{n-1,n-1}x_{n-1} \\ x_{n-2} &:= l_{n-2,1}x_1 + l_{n-2,2}x_2 + l_{n-2,3}x_3 + \ldots + l_{n-2,n-2}x_{n-2} \\ &\vdots \quad \dots \quad \dots \quad \dots \quad \dots \quad \dots \quad \dots \quad \dots \quad \dots \\ &\vdots \quad \dots \quad \dots \quad \dots \quad \dots \quad \dots \quad \dots \quad \dots \quad \dots \\ &\vdots \quad \dots \quad \dots \quad \dots \quad \dots \quad \dots \quad \dots \quad \dots \quad \dots \\ x_3 &:= l_{31}x_1 + l_{32}x_2 + l_{33}x_3 \\ x_2 &:= l_{21}x_1 + l_{22}x_2 \\ x_1 &:= l_{11}x_1 \end{aligned}$$

Now, let $\acute{A}$ and $\acute{V}$ denote the modified form of the matrix A and vector V (of variables) after performing any operation. Then, after each operation, it must preserve and compute the original mapping effectively *i.e.*

$$AV = \acute{A}\acute{V}$$

If T denotes the transformation matrix then

$$\begin{aligned} \acute{A} &= AT \\ \acute{V} &= T^{-1}V \\ \implies \acute{A}\acute{V} &= A(TT^{-1})V = AV \end{aligned}$$

□

We notice that, an assignment $x_i := \alpha x_i + f$, repeated k times can be written as a single assignment of the form.

$$x_i := \alpha^k x_i + f\left(\alpha^{k-1} + \alpha^{k-2} + \ldots + \alpha^2 + \alpha^1 + \alpha^0\right),\ \alpha = 1$$

It enables us to modify the algorithm to avoid from repeated similar assignments. So, we introduce the following modifications.

5.1.1 Modifications:

Case 2: If $a_{ii} > a_{ij}$, $for\, j > i$, then the program will perform the operation $C_i := C_i - \beta_1 C_j$, and x_j will be modified to $x_j := \alpha x_j + \beta_1 f$, where f does not depend on x_j. C_i denotes the ith column of matrix A and $\beta_1 = trunc(a_{ii}/a_{ij})$. The function $trunc$ returns integer quotient.

Case 3: If $a_{ij} \geq a_{ii}$, $for\, j > i$, then the program will perform the operation $C_j := C_j - \beta_2 C_i$, and x_i will be modified to $x_i := \alpha x_i + \beta_2 f$, where f does not depend on x_i and $\beta_1 = trunc(a_{ij}/a_{ii})$. The case $a_{ij} = 0$, can be handled by taking $\beta_2 = 1$

"We estimate an upper bound over the number of assignments with the help of famous result presented by Gabriel Lamé [Lam44]".

Suppose that γ denotes the total number of assignments involve in computing linear mapping E, then $\gamma \leq 5(n-1)d + n$, where d denotes the number of digits in the entry $a(i, j)$ of the matrix A such that $i \leq j$, but $i \neq n$, Matrix A is the matrix of coefficients of mapping E.

5.1.2 Explanation and construction of assignments:

We explain the construction of assignments by the following Example 15 and will discuss step by step in Example 16.

Example 15. *Suppose that E be the mapping with integer coefficients as defined by*

$$E\,(x_1,\, x_2) \longrightarrow (13x_1 + 21x_2,\, 21x_1 + 34x_2)$$

Let the matrix M_c represents the coefficients of mapping E.

$$M_c := \begin{bmatrix} 13 & 21 \\ 21 & 34 \end{bmatrix}$$

We generate sequence of linear assignments by performing column/row operations on matrix M_c in such a way that it results with integer entries (excluding division operation). In fact, the square matrix M_c is transformed into triangular matrix, at the completion of all operations. In Table 5.1, we show the steps involve in generating sequence of linear assignments, that involve in computing linear mapping E.

Table 5.1: Example 15

Generating linear assignments			
Matrix(input)	*Operation*	*Assignment*	*Matrix(output)*
$\begin{bmatrix}13 & 21\\21 & 34\end{bmatrix}$	$C_2 - C_1$	$x_1 := x_1 + x_2$	$\begin{bmatrix}13 & 8\\21 & 13\end{bmatrix}$
$\begin{bmatrix}13 & 8\\21 & 13\end{bmatrix}$	$C_1 - C_2$	$x_2 := x_1 + x_2$	$\begin{bmatrix}5 & 8\\8 & 13\end{bmatrix}$
$\begin{bmatrix}5 & 8\\8 & 13\end{bmatrix}$	$C_2 - C_1$	$x_1 := x_1 + x_2$	$\begin{bmatrix}5 & 3\\8 & 5\end{bmatrix}$
$\begin{bmatrix}5 & 3\\8 & 5\end{bmatrix}$	$C_1 - C_2$	$x_2 := x_1 + x_2$	$\begin{bmatrix}2 & 3\\3 & 5\end{bmatrix}$
$\begin{bmatrix}2 & 3\\3 & 5\end{bmatrix}$	$C_2 - C_1$	$x_1 := x_1 + x_2$	$\begin{bmatrix}2 & 1\\3 & 2\end{bmatrix}$
$\begin{bmatrix}2 & 1\\3 & 2\end{bmatrix}$	$C_1 - C_2$	$x_2 := x_1 + x_2$	$\begin{bmatrix}1 & 1\\1 & 2\end{bmatrix}$
$\begin{bmatrix}1 & 1\\1 & 2\end{bmatrix}$	$C_2 - C_1$	$x_1 := x_1 + x_2$	$\begin{bmatrix}1 & 0\\1 & 1\end{bmatrix}$

In Table 5.1, first column presents input matrices and second column presents column operations performed on these matrices. The corresponding assignments constructed are in third column while matrices (modified) after performing operations are presented in last column of the table. Instead of column operations, row operations could be performed and assignments could be constructed accordingly.

We terminate the operations, when square matrix M_c is transformed into triangular matrix, and we generate, the next part of sequence of assignments, directly from triangular matrix, writing from last row to first as given below.

$$\begin{bmatrix}1 & 0\\1 & 1\end{bmatrix} \longrightarrow \begin{cases} x_2 := x_1 + x_2 \\ x_1 := x_1 \end{cases}$$

If we ignore the last assignment, it will not affect the computation and we will have a sequence of eight linear assignments in total.

Computing inverse mappings:

Generate the set of linear assignments that compute inverse mapping E^{-1}, directly from the set of linear assignments that computes mapping E, by inverting and rewriting from the last assignment to the first as given below.

Table 5.2: Example 15

Generating linear assignments for E^{-1}			
Input	*Inverted*	*Evaluated*	*Coefficients*
$x_2 := x_1 + x_2$	$x_2 := -x_1 + x_2$	$x_2 := -x_1 + x_2$	$\begin{bmatrix}-1 & 1\end{bmatrix}$
$x_1 := x_1 + x_2$	$x_1 := x_1 - x_2$	$x_1 := 2x_1 - x_2$	$\begin{bmatrix}2 & -1\end{bmatrix}$
$x_2 := x_1 + x_2$	$x_2 := -x_1 + x_2$	$x_2 := -3x_1 + 2x_2$	$\begin{bmatrix}-3 & 2\end{bmatrix}$
$x_1 := x_1 + x_2$	$x_1 := x_1 - x_2$	$x_1 := 5x_1 - 3x_2$	$\begin{bmatrix}5 & -3\end{bmatrix}$
$x_2 := x_1 + x_2$	$x_2 := -x_1 + x_2$	$x_2 := -8x_1 + 5x_2$	$\begin{bmatrix}-8 & 5\end{bmatrix}$
$x_1 := x_1 + x_2$	$x_1 := x_1 - x_2$	$x_1 := 13x_1 - 8x_2$	$\begin{bmatrix}13 & -8\end{bmatrix}$
$x_2 := x_1 + x_2$	$x_2 := -x_1 + x_2$	$x_2 := -21x_1 + 13x_2$	$\begin{bmatrix}-21 & 13\end{bmatrix}$
$x_1 := x_1 + x_2$	$x_1 := x_1 - x_2$	$x_1 := 34x_1 - 21x_2$	$\begin{bmatrix}34 & -21\end{bmatrix}$

Second column of the Table 5.2 presents set of linear assignments that compute the inverse mapping E^{-1} of E defined by

$$E^{-1}(x_1,\, x_2) \longrightarrow (34x_1 - 21x_2,\, -21x_1 + 13x_2)$$

Consequently, we obtain inverse of matrix A as given below.

$$A^{-1} := \begin{bmatrix} 34 & -21 \\ -21 & 13 \end{bmatrix}$$

$$A * A^{-1} = \begin{bmatrix}13 & 21\\ 21 & 34\end{bmatrix} * \begin{bmatrix} 34 & -21 \\ -21 & 13 \end{bmatrix} = \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}$$

Next, we give another example that explains the process of construction of assignments step by step.

Example 16. *Consider a linear mapping defined as*

$$E(x_1,\, x_2,\, x_3) \longrightarrow (5x_1 - 3x_2 + 5x_3,\, 3x_1 - 7x_3,\, 4x_1 + 8x_2 + 13x_3)$$

We are interested to compute the given mapping by sequence of linear assignments such that the coefficients of these assignments are integers. Let matrix 'A' presents coefficients of given mapping E.

$$A := \begin{bmatrix} 5 & -3 & 5 \\ 3 & 0 & -7 \\ 4 & 8 & 13 \end{bmatrix}$$

We compute mapping E by applying the procedure, keeping in mind different cases, as described in Theorem 3.

Now, notice that $a_{12} < 0$, therefore, we are in the first case and we perform the operation $C_2 := -C_2$. The linear assignment corresponding to this operation is

$$x_2 := -x_2$$

and matrix A will be modified to the form

$$A := \begin{bmatrix} 5 & 3 & 5 \\ 3 & 0 & -7 \\ 4 & -8 & 13 \end{bmatrix}$$

Now, because $a_{11} = 5 > a_{12} = 3$, therefore, we are in the second case and we perform the operation $C_1 := C_1 - C_2$. The linear assignment corresponding to this operation is

$$x_2 := x_1 + x_2$$

and the matrix A will be modified.

$$A := \begin{bmatrix} 2 & 3 & 5 \\ 3 & 0 & -7 \\ 12 & -8 & 13 \end{bmatrix}$$

Now, observe that $a_{12} = 3 \geq a_{11} = 2$ therefore, we are in the third case and we perform the operation $C_2 := C_2 - C_1$. The linear assignment corresponding to this operation is

$$x_1 := x_1 + x_2$$

and the matrix A will be modified.

$$A := \begin{bmatrix} 2 & 1 & 5 \\ 3 & -3 & -7 \\ 12 & -20 & 13 \end{bmatrix}$$

Since $a_{11} = 2 > a_{12} = 1$, therefore, we perform the operation $C_1 := C_1 - C_2$. The linear assignment corresponding to this operation is $x_2 := x_1 + x_2$ and the matrix A will be modified.

$$A := \begin{bmatrix} 1 & 1 & 5 \\ 6 & -3 & -7 \\ 32 & -20 & 13 \end{bmatrix}$$

Since $a_{12} = 1 \geq a_{11} = 1$, therefore, we perform the operation $C_2 := C_2 - C_1$. The linear assignment corresponding to this operation is $x_1 := x_1 + x_2$ and the matrix A will be modified.

$$A := \begin{bmatrix} 1 & 0 & 5 \\ 6 & -9 & -7 \\ 32 & -52 & 13 \end{bmatrix}$$

Observe that a_{12} became zero, We will continue this process until $a_{13} = 0$.
Since $a_{13} = 5 \geq a_{11} = 1$, we perform the operation $C_3 := C_3 - C_1$. The linear assignment corresponding to this operation is $x_1 := x_1 + x_3$. The matrix A will be modified.

$$A := \begin{bmatrix} 1 & 0 & 4 \\ 6 & -9 & -13 \\ 32 & -52 & -19 \end{bmatrix}$$

Since $a_{13} = 4 \geq a_{11} = 1$, we perform the operation $C_3 := C_3 - C_1$. The linear assignment corresponding to this operation is $x_1 := x_1 + x_3$. The matrix A will be modified.

$$A := \begin{bmatrix} 1 & 0 & 3 \\ 6 & -9 & -19 \\ 32 & -52 & -51 \end{bmatrix}$$

Since $a_{13} = 3 \geq a_{11} = 1$, we perform the operation $C_3 := C_3 - C_1$. The linear assignment corresponding to this operation is $x_1 := x_1 + x_3$. The matrix A will be modified.

$$A := \begin{bmatrix} 1 & 0 & 2 \\ 6 & -9 & -25 \\ 32 & -52 & -83 \end{bmatrix}$$

Since $a_{13} = 2 \geq a_{11} = 1$, we perform the operation $C_3 := C_3 - C_1$. The linear assignment corresponding to this operation is $x_1 := x_1 + x_3$. The matrix A will be modified.

$$A := \begin{bmatrix} 1 & 0 & 1 \\ 6 & -9 & -31 \\ 32 & -52 & -115 \end{bmatrix}$$

Since $a_{13} = 1 \geq a_{11} = 1$, we perform the operation $C_3 := C_3 - C_1$. The linear assignment corresponding to this operation is $x_1 := x_1 + x_3$. The matrix A will be modified.

$$A := \begin{bmatrix} 1 & 0 & 0 \\ 6 & -9 & -37 \\ 32 & -52 & -147 \end{bmatrix}$$

Now the first column will remain unchanged throughout the next operations, we will focus on second diagonal entry.
Since $a_{23} = -37 < 0$, we perform the operation $C_3 := -C_3$. The linear

assignment corresponding to this operation is $x_3 := -x_3$. *The matrix* A *will be modified.*

$$A := \begin{bmatrix} 1 & 0 & 0 \\ 6 & -9 & 37 \\ 32 & -52 & 147 \end{bmatrix}$$

Since $a_{22} = -9 < 0$, *we perform the operation* $C_2 := -C_2$. *The linear assignment corresponding to this operation is* $x_2 := -x_2$. *The matrix* A *will be modified.*

$$A := \begin{bmatrix} 1 & 0 & 0 \\ 6 & 9 & 37 \\ 32 & 52 & 147 \end{bmatrix}$$

Since $a_{23} = 37 \geq a_{22} = 9$, *we perform the operation* $C_3 := C_3 - C_2$. *The linear assignment corresponding to this operation is* $x_2 := x_2 + x_3$. *The matrix* A *will be modified.*

$$A := \begin{bmatrix} 1 & 0 & 0 \\ 6 & 9 & 28 \\ 32 & 52 & 95 \end{bmatrix}$$

since $a_{23} = 28 \geq a_{22} = 9$, *we perform the operation* $C_3 := C_3 - C_2$. *The linear assignment corresponding to this operation is* $x_2 := x_2 + x_3$. *The matrix* A *will be modified.*

$$A := \begin{bmatrix} 1 & 0 & 0 \\ 6 & 9 & 19 \\ 32 & 52 & 43 \end{bmatrix}$$

Since $a_{23} = 19 \geq a_{22} = 9$, *we perform the operation* $C_3 := C_3 - C_2$. *The linear assignment corresponding to this operation is* $x_2 := x_2 + x_3$. *The matrix* A *will be modified.*

$$A := \begin{bmatrix} 1 & 0 & 0 \\ 6 & 9 & 10 \\ 32 & 52 & -9 \end{bmatrix}$$

Since $a_{23} = 10 \geq a_{22} = 9$, *we perform the operation* $C_3 := C_3 - C_2$. *The linear assignment corresponding to this operation is* $x_2 := x_2 + x_3$. *The matrix* A *will be modified.*

$$A := \begin{bmatrix} 1 & 0 & 0 \\ 6 & 9 & 1 \\ 32 & 52 & -61 \end{bmatrix}$$

Since $a_{22} = 9 > a_{23} = 1$, *we perform the operation* $C_2 := C_2 - C_3$. *The linear assignment corresponding to this operation is* $x_3 := x_2 + x_3$. *The matrix* A *will be modified.*

$$A := \begin{bmatrix} 1 & 0 & 0 \\ 6 & 8 & 1 \\ 32 & 113 & -61 \end{bmatrix}$$

Since $a_{22} = 8 > a_{23} = 1$, *we perform the operation* $C_2 := C_2 - C_3$. *The linear assignment corresponding to this operation is* $x_3 := x_2 + x_3$. *The matrix* A *will be modified.*

$$A := \begin{bmatrix} 1 & 0 & 0 \\ 6 & 7 & 1 \\ 32 & 174 & -61 \end{bmatrix}$$

Since $a_{22} = 7 > a_{23} = 1$, *we perform the operation* $C_2 := C_2 - C_3$. *The linear assignment corresponding to this operation is* $x_3 := x_2 + x_3$. *The matrix* A *will be modified.*

$$A := \begin{bmatrix} 1 & 0 & 0 \\ 6 & 6 & 1 \\ 32 & 235 & -61 \end{bmatrix}$$

Since $a_{22} = 6 > a_{23} = 1$, *we perform the operation* $C_2 := C_2 - C_3$. *The linear assignment corresponding to this operation is* $x_3 := x_2 + x_3$. *The matrix* A *will be modified.*

$$A := \begin{bmatrix} 1 & 0 & 0 \\ 6 & 5 & 1 \\ 32 & 296 & -61 \end{bmatrix}$$

Since $a_{22} = 5 > a_{23} = 1$, *we perform the operation* $C_2 := C_2 - C_3$. *The linear assignment corresponding to this operation is* $x_3 := x_2 + x_3$. *The matrix* A *will be modified.*

$$A := \begin{bmatrix} 1 & 0 & 0 \\ 6 & 4 & 1 \\ 32 & 357 & -61 \end{bmatrix}$$

Since $a_{22} = 4 > a_{23} = 1$, *we perform the operation* $C_2 := C_2 - C_3$. *The linear assignment corresponding to this operation is* $x_3 := x_2 + x_3$. *The matrix* A *will be modified.*

$$A := \begin{bmatrix} 1 & 0 & 0 \\ 6 & 3 & 1 \\ 32 & 418 & -61 \end{bmatrix}$$

Since $a_{22} = 3 > a_{23} = 1$, *we perform the operation* $C_2 := C_2 - C_3$. *The linear assignment corresponding to this operation is* $x_3 := x_2 + x_3$. *The matrix* A *will be modified.*

$$A := \begin{bmatrix} 1 & 0 & 0 \\ 6 & 2 & 1 \\ 32 & 479 & -61 \end{bmatrix}$$

Since $a_{22} = 2 > a_{23} = 1$, *we perform the operation* $C_2 := C_2 - C_3$. *The linear assignment corresponding to this operation is* $x_3 := x_2 + x_3$. *The matrix* A *will be modified.*

$$A := \begin{bmatrix} 1 & 0 & 0 \\ 6 & 1 & 1 \\ 32 & 540 & -61 \end{bmatrix}$$

Since $a_{23} = 1 \geq a_{22} = 1$, we perform the operation $C_3 := C_3 - C_2$. The linear assignment corresponding to this operation is $x_2 := x_2 + x_3$. The matrix A will be modified.

$$A := \begin{bmatrix} 1 & 0 & 0 \\ 6 & 1 & 0 \\ 32 & 540 & -601 \end{bmatrix}$$

The matrix A is converted into lower triangular matrix and the last three assignments will be written directly from this lower triangular matrix. These three assignments are:

$$\begin{cases} x_3 & := 32x_1 + 540x_2 - 601x_3 \\ x_2 & := 6x_1 + x_2 \\ x_1 & := x_1 \end{cases}$$

Possibility of computing inverse mappings:

IDC for inverse mappings over integers is not always possible. We illustrate this by the following example

Example 17. *Consider a mapping E*

$$E\,(x_1,\, x_2,\, x_3) \longrightarrow (2x_1 + 3x_2 + 5x_3,\, 3x_1 + 4x_2 - 7x_3,\, 8x_2 + 13x_3)$$

The mapping E can be computed by the following sequence of assignments.

$$\textit{Assignments} \begin{cases} x_1 := x_1 + x_2 \\ x_2 := 2x_1 + x_2 \\ x_2 := x_1 + x_2 \\ x_1 := -x_1 \\ x_1 := x_1 + x_2 \\ x_1 := x_1 + 5x_3 \\ x_3 := -x_3 \\ x_2 := x_2 + 7x_3 \\ x_3 := 24x_1 - 16x_2 + 219x_3 \\ x_2 := x_2 \\ x_1 := x_1 \end{cases}$$

The matrix of coefficients A for the mapping E can be written as

$$A := \begin{bmatrix} 2 & 3 & 5 \\ 3 & 4 & -7 \\ 0 & 8 & 13 \end{bmatrix}$$

and

$$A^{-1} := \frac{1}{219} \begin{bmatrix} 108 & 1 & -41 \\ -39 & 26 & 29 \\ 24 & -16 & -1 \end{bmatrix}$$

Notice that $\frac{1}{219} \notin \mathbb{Z}$ *and the assignment*

$$x_3 := 24x_1 - 16x_2 + 219x_3$$

is not invertible. Hence, the inverse mapping E^{-1} *is not always computable by inverting and rewriting the sequence of assignments that computes mapping* E*.*

5.2 Investigating bounds for the number of assignments:

We are interested in determining the minimum number of assignments required to compute given mapping by *"In Situ Design of Computation (IDC)"* with integer coefficients. We proceed by developing relations between mappings that help to find minimum number of assignments and investigate by finding different counter examples. Based on, one of such counter examples, we provide the following theorem.

Theorem 4. *There exists a linear mapping*

$$E : (x,\, y) \longrightarrow (mx + ny,\, px + qy)$$

with $m,\, n,\, p,\, q \in \mathbb{Z}$*, that cannot be computed by a sequence of at most* 6 *linear assignments*

$$\left.\begin{array}{l} x := ax + by \\ y := cx + dy \\ x := ex + fy \\ y := gx + hy \\ x := ix + jy \\ y := kx + ly \end{array}\right\}$$

where $a,\, b,\, c,\, d,\, e,\, f,\, g,\, h,\, i,\, j,\, k,$ *and* $l \in \mathbb{Z}$*.*

Proof. Let E be a linear mapping $E(x,\, y) \longrightarrow (461x + 286y,\, 353x + 219y)$, whose coefficients can be viewed as a matrix given below.

$$\begin{bmatrix} 461 & 286 \\ 353 & 219 \end{bmatrix} = A(say)$$

We establish a system of four equations (by evaluating assignments), as given below:

$$\left\{\begin{array}{l} 461 = a(ie + ifc + jge + jgfc + jhc) \\ 286 = ieb + ifcb + ifd + jgeb + jgfcb + jgfd + jhcb + jhd \\ 353 = 461k + lgea + lgfca + lhca \\ 219 = 286k + lgeb + lgfcb + lgfd + lhcb + lhd \end{array}\right. \tag{2}$$

$$\text{determinant (A)} = mq - pn = aedihl$$

$\Longrightarrow$ The product of variables a, d, e, h, i, and l must be equal to determinant of matrix A. Since, determinant of matrix A is equal to 1. Therefore,

$$a * e * d * h * i * l = 1$$

and each of six variables take values from the set $\{1, -1\}$. Notice that, there are

$$P(6,\, 2) + 2 = 32$$

cases of assigning values to a, d, e, h, i, and l.

Let us start by considering the case with values

$$a = 1,\, e = 1,\, d = 1,\, h = 1,\, i = 1,\, l = 1$$

It gives a set of equations:

$$\begin{cases} 460 = fc + jg + jgfc + jc \\ 286 = 461b + f + jgf + j \\ 353 = 461k + g + gfc + c \\ 218 = 286k + 353b - 461bk + gf \end{cases} \tag{3}$$

We observe that there are three general solutions for system 3 of equations: First possible solution is of the form:

$$\left.\begin{aligned} c = \frac{461}{j},\, j = j,\, f = f,\, b = -\frac{1}{461}j + \frac{286}{461} \\ k = \frac{1}{461}\frac{-460j + 461f + 353j^2}{j^2},\, g = -\frac{1}{j} \end{aligned}\right\} \quad \text{(first)}$$

Second possible solution is of the form:

$$\left.\begin{aligned} j = j,\, g = \frac{460}{j},\, k = \frac{1}{461}\frac{-460 + 353j}{j} \\ b = -f - \frac{1}{461}j + \frac{286}{461},\, f = f,\, c = 0 \end{aligned}\right\} \quad \text{(second)}$$

The third possible solution is of the form:

$$\left.\begin{aligned} j = j,\, g = g,\, k = -\frac{1}{461}\frac{461g + c - 353 - 353jg}{1 + jg} \\ c = c,\, b = \frac{1}{461}\frac{jg - 460 + 286c}{c},\, f = -\frac{-460 + jg + jc}{c(1 + jg)} \end{aligned}\right\} \quad \text{(third)}$$

Consider the first possible solution:

$$\text{Let} \quad g = -\frac{1}{j} \in \mathbb{Z} \implies j = 1 \; or \; -1$$

Now, If $j = 1$ then

$$g = -1 \in \mathbb{Z},\ \text{but}\ b = -\frac{1}{461} + \frac{286}{461} = \frac{285}{461} \notin \mathbb{Z}$$

Similarly, If $j = -1$ then

$$g = 1 \in \mathbb{Z}, \text{ but } b = \frac{1}{461} + \frac{286}{461} = \frac{287}{461} \notin \mathbb{Z}$$

Therefore integer solution is not possible for the solution first. In the second possible solution:

$$g = \frac{460}{j}, k = \frac{1}{461}\frac{-460 + 353j}{j}$$

$$k \text{ can be written as } k = \frac{-g + 353}{461}$$

Observe that if $k \in \mathbb{Z}$ then $-g + 353 \geq 461$

$$\implies g \leq 353 - 461 \implies g \leq -108$$

'g' is a divisor of 460 and it takes value from the set $\{-115, -230, -460\}$ because $g \leq -108$. But for any of these values $\{-115, -230, -460\}$ $k \notin \mathbb{Z}$. Notice that,

$$if\ g = -230\ then\ k = \frac{230 + 353}{461} = \frac{583}{461} \notin \mathbb{Z}$$

$$if\ g = -115\ then\ k = \frac{115 + 353}{461} = \frac{468}{461} \notin \mathbb{Z}$$

$$if\ g = -460\ then\ k = \frac{460 + 353}{461} = \frac{813}{461} \notin \mathbb{Z}$$

Therefore integer solution is not possible for second solution second. Now, in the third possible solution:

$$b = \frac{1}{461}\frac{jg - 460 + 286c}{c} \text{ and } f = -\frac{-460 + jg + jc}{c(1 + jg)}$$

Since c divides both $jg - 460 + 286c$ and $-460 + jg + jc$, therefore, if $b, f \in \mathbb{Z}$ then there exists some $k_1, k_2 \in \mathbb{Z}$ such that

$$460 - jg - jc = ck_1 \tag{5.1}$$

and

$$460 - jg = c(286 - 461k_2) \tag{5.2}$$

By (5.1) and (5.2)

$$c(j + k_1) = c(286 - 461k_2)$$

$$\implies j + k_1 = 286 - 461k_2$$

$$\implies k_2 = -\frac{-286 + k_1 + j}{461}$$

If $k_2 \in \mathbb{Z}$ then $-286 + k_1 + j \geq 461$

$$\implies k_1 + j \geq 747 \tag{5.3}$$

$$c = \frac{460 - jg}{j + k_1} \quad \text{by equation (5.1)}$$

and if $c \in \mathbb{Z}$ then

$$j + k_1 \leq 460 - jg \tag{5.4}$$

By (5.3) and (5.4), we can write

$$460 - jg = 747 \Longrightarrow jg = -287$$

$\Longrightarrow$ For j and g to be integers, they must be factors of 287 and the possible factors of 287 are

$$\{\pm 1, \pm 7, \pm 41, \pm 287\}$$

Let, for example, $g = -287$ then $j = 1$

$$\Longrightarrow k_1 = 746 \Longrightarrow k_2 = -1 \Longrightarrow c = 1$$

$$But,\ then,\ f = -\frac{-460 - 287 + 1}{1 - 287} = -\frac{34}{13} \notin \mathbb{Z}$$

Thus, for all divisors of 287, $f \notin \mathbb{Z}$. In a similar way, it can be proved for other 31 possibilities of assigning values $\{1, -1\}$ to variables a, d, e, h, i, and l. □

5.2.1 An approach based on Fibonacci numbers

To investigate the minimum number of assignments required to perform *"In Situ Design of Computation (IDC)"* for mappings over integers, we establish relations, keeping Fibonacci numbers, as coefficients of the mapping.

Definition 4. *We define a Fibonacci-like sequence as*

$$F_n = F_{n-1} + F_{n-2}$$

where $F_0 = F_1 = 1$.

Definition 5. *We define a Fibonacci-like matrix to be a matrix of the form*

$$\begin{bmatrix} F_{n-1} & F_n \\ F_n & F_{n+1} \end{bmatrix}$$

and the following relations.

$$R_1: \quad \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^{n+1} = \begin{bmatrix} F_{n-1} & F_n \\ F_n & F_{n+1} \end{bmatrix}$$

$$R_2: \quad \begin{bmatrix} F_2 & F_3 \\ F_3 & F_4 \end{bmatrix} * \begin{bmatrix} F_{4n+1} & F_{4n+2} \\ F_{4n+2} & F_{4n+3} \end{bmatrix} = \begin{bmatrix} F_{4n+4} & F_{4n+5} \\ F_{4n+5} & F_{4n+6} \end{bmatrix}$$

Theorem 5. *Let* $E_n : (x, y) \longrightarrow (F_{n-1}x + F_n y,\ F_n x + F_{n+1} y)$ *be the mapping on* $\mathbb{Z}^2$*, where* F_n *is the Fibonacci number. Then the mapping* E_{4k+2} *is computed with* $2k + 2$ *number of assignments, where* $k = 0, 1, 2, \cdots, n$.

Proof. For $n = 2$, $E_2 = (F_1x + F_2y,\ F_2x + F_3y)$ and the matrix of coefficients is

$$A_2 = \begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix}$$

$$\text{The assignments for } A_2 \rightarrow \begin{cases} x := x + 2y \\ y := 2x - y \end{cases}$$

$$A_3 = \begin{bmatrix} 2 & 3 \\ 3 & 5 \end{bmatrix}$$

$$\text{The assignments for } A_3 \rightarrow \begin{cases} x := x + 2y \\ y := 3x - y \\ x := -x + y \end{cases}$$

Now,

$$A_3 = \begin{bmatrix} 2 & 3 \\ 3 & 5 \end{bmatrix} * A_2 = \begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix} = A_6 = \begin{bmatrix} 8 & 13 \\ 13 & 21 \end{bmatrix} \quad \text{by } R_2.$$

We obtain sequence of assignments required to compute E_6 as follows. In fact, we pack together the assignments require to compute E_3 and the assignments require to compute E_2.

$$\text{The assignments for } A_6 \rightarrow \begin{cases} x := x + 2y \\ y := 3x - y \\ x := -x + y \\ x := x + 2y \\ y := 2x - y \end{cases} \rightarrow \begin{cases} x := x + 2y \\ y := 3x - y \\ x := -x + 3y \\ y := 2x - y \end{cases}$$

$$\text{Similarly, since } A_3 = \begin{bmatrix} 2 & 3 \\ 3 & 5 \end{bmatrix} * A_6 = \begin{bmatrix} 8 & 13 \\ 13 & 21 \end{bmatrix} = \begin{bmatrix} 55 & 89 \\ 89 & 144 \end{bmatrix} = A_{10}$$

Therefore,

$$\text{The assignments for } A_{10} \rightarrow \begin{cases} x := x + 2y \\ y := 3x - y \\ x := -x + y \\ x := x + 2y \\ y := 3x - y \\ x := -x + 3y \\ y := 2x - y \end{cases} \rightarrow \begin{cases} x := x + 2y \\ y := 3x - y \\ x := -x + 3y \\ y := 3x - y \\ x := -x + 3y \\ y := 2x - y \end{cases}$$

Continuing in this way, we have, $A_3 * A_{4k+2} = A_{4k+6}$

and

$$\text{The assignments for } A_{4k+6} \rightarrow \left\{\begin{array}{l} x := x+2y \\ y := 3x-y \\ x := -x+y \\ x := x+2y \\ y := 3x-y \\ x := -x+3y \\ \quad \vdots \\ y := 3x-y \\ x := -x+3y \\ y := 2x-y \end{array}\right. = \left\{\begin{array}{l} x := x+2y \\ y := 3x-y \\ x := -x+3y \\ \quad \vdots \\ y := 3x-y \\ x := -x+3y \\ y := 2x-y \end{array}\right.$$

Hence, the number of assignments required to compute mapping whose coefficients are presenting by matrix A_{4k+2} is $2k+2$. □

Proposition 2. *The linear mapping*

$$E : \mathbb{Z}^m \longrightarrow \mathbb{Z}^m,\ m > 2$$

with Fibonacci numbers as coefficients, defined by

$$E : (x_1, x_2, x_3, \cdots, x_n) = \begin{bmatrix} F_1x_1 & + & F_2x_2 & + & F_3x_3 & +\ldots & + & F_mx_n \\ F_2x_1 & + & F_3x_2 & + & F_4x_3 & +\ldots & + & F_{m+1}x_n \\ F_3x_1 & + & F_4x_2 & + & F_5x_3 & +\ldots & + & F_{m+2}x_n \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \vdots \\ F_nx_1 & + & F_{n+1}x_2 & + & F_{n+2}x_3 & +\ldots & + & F_{2n-1}x_n \end{bmatrix}$$

such that $F_n := F_{n-1} + F_{n-2}$, $n \in \mathbb{Z}$, is computed with $m+2$ number of linear assignments.

Proof. The given mapping E can be written as $X := AX$ where A is a matrix such that whose entries satisfy the relation $F_n := F_{n-1} + F_{n-2}$, $\forall n \in \mathbb{Z}$. we construct assignments as given below:

$$\text{First two assignments } \rightarrow \left\{\begin{array}{l} x_1 := x_1 + x_2 + 2x_3 + 3x_4 + 5x_5 + \ldots \\ x_2 := x_1 + x_2 + x_3 + 2x_4 + 3x_5 + \ldots \end{array}\right.$$

$$\text{Next two assignments } \rightarrow \left\{\begin{array}{l} x_3 := F_1x_1 + F_2x_2 \\ x_4 := x_3 + (F_2 - F_1)\,x_1 + F_1x_2 \end{array}\right.$$

$$\text{Intermediate assignments } \rightarrow \left\{\begin{array}{l} x_5 := x_3 + x_4 \\ x_6 := x_4 + x_5 \\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots \\ x_n := x_{n-2} + x_{n-1} \end{array}\right. .$$

$$\text{Last two assignments } \rightarrow \left\{\begin{array}{l} x_1 := x_3 - (F_2 - F_1)\,x_1 + F_1x_2 \\ x_2 := x_3 - x_1 \end{array}\right.$$

This sequence of assignments compute mapping E effectively, and the number of these assignments are $n+2$. □

Next, we prove an identity that relates determinant of matrix of coefficients to product of coefficients of assignments.

5.2.2 Identity (coefficient linkage):

The sequence of linear assignments (with integer coefficients) that compute mapping E has an interesting property to compute determinant of matrix M_c corresponding to mapping E. To explore this property, we prove the following identity.

Theorem 6. *Let $E : \mathbb{Z}^2 \longrightarrow \mathbb{Z}^2$ be a linear mapping defined as*

$$E(x, y) \longrightarrow (mx + ny,\ px + qy)$$

with integral coefficients m, n, p, $q \in \mathbb{Z}$. Let E be computable by a sequence of linear assignments

$$\left.\begin{array}{l} x := a_1x + b_1y \\ y := b'_1x + a'_1y \\ x := a_2x + b_2y \\ y := b'_2x + a'_2y \\ \vdots := \quad \vdots \quad \vdots \\ x := a_kx + b_ky \\ y := b'_kx + a'_ky \end{array}\right\}$$

where a_i, a'_i, b_i, $b'_i \in \mathbb{Z}$ are integral coefficients for each $i = 1, \ldots, k$. Then the identity

$$mq - pn = a_1a'_1 \ldots a_ka'_k \qquad \textit{holds.} \tag{5.5}$$

Proof. Proof by induction on k.
Base: For $k = 1$ we have $x = a_1x + b_1y$ and $y = a_1b'_1x + (b_1b'_1 + a'_1)y$, where $m = a_1$, $n = b_1$, $p = a_1b'_1$, and $q = b_1b'_1 + a'_1$.
Hence we have $mq - pn = a_1(b_1b'_1 + a'_1) - a_1b_1b'_1 = a_1a'_1$, that satisfies the required identity 5.5. Step: Let the identity 5.5 holds for some $k - 1$, we will prove it for k.
We have $E(x, y) \longrightarrow (mx + ny,\ px + qy)$ computed by the aforementioned linear assignment sequence of length $k - 1$. To this we apply the step k with $x := a_kx + b_ky$ followed by $y := b'_kx + a'_ky$. After substitution we obtain $x = a_kx + b_ky$ and $y = a_kb'_kx + (b_kb'_k + a'_k)y$. We substitute $x := mx + ny$ and $y := px + qy$ respectively in the right-hand sides, and obtain after some simple manipulation, we have

$$\begin{aligned} x &= (a_km + b_kp)x + (a_kn + b_kq)y = m'x + n'y \\ y &= (a_kb'_km + b_kb'_kp + a'_kp)x + (a_kb'_kn + b_kb'_kq + a'_kq)y = p'x + q'y. \end{aligned}$$

Now we have new values for the coefficients m, n, p and q denoted by m', n', p' and q' which are equal to $m' = a_km + b_kp$, $n' = a_kn + b_kq$, $p' = a_kb'_km + b_kb'_kp + a'_kp$, $q' = a_kb'_kn + b_kb'_kq + a'_kq$. The expression $m'q' - p'n'$, after a long and

tedious multiplication and elimination calculus, is equal to $(mq - pn)a_k a'_k$. By assumption hypothesis we have $mq - pn = a_1 a'_1 \ldots a_{k-1} a'_{k-1}$, what proves the required identity. □

Example 18. *Let $E : \mathbb{Z}^2 \longrightarrow \mathbb{Z}^2$ be a linear mapping defined as*

$$E(x, y) \longrightarrow (33x + 307y, 103x + 610y)$$

The coefficients of mapping can be represented by matrix as given below.

$$A = \begin{bmatrix} 33 & 307 \\ 103 & 610 \end{bmatrix}$$

$$\textit{determinant } (A) = -11491 \tag{5.6}$$

Mapping E can be computed by the following sequence of linear assignments

$$\textit{Assignments} \begin{cases} x := x + 9y \\ y := 3x + y \\ x := x + 3y \\ y := 2x + y \\ x := x + y \\ y := 8012x - 11491y \end{cases}$$

$$\textit{Product of coefficients } a_i s = -11491 \tag{5.7}$$

The identity (Theorem 6) helps in determining whether a sequence (of linear assignments) that computes the inverse mapping, exists or not. For example, for the mapping $E : \mathbb{Z}^2 \longrightarrow \mathbb{Z}^2$, a sequence that computes E^{-1}, exists if R.H.S of identity 5.5 becomes numerically equal to 1.

Chapter 6

In Situ Design of Computation for Boolean Mappings

Contents

6.1 Boolean Mappings 78

6.2 Computing Bijective Boolean Mappings . . . 79

6.2.1 Explanation 81

6.2.2 Linearity Property 83

6.3 Computing General Boolean Mappings 84

6.3.1 Simple Mappings 85

6.3.2 Decomposing general boolean mappings 86

6.4 IDC for Polynomials over GF(2) 91

6.4.1 A First Tool 92

6.4.2 Explanation and Construction 95

Decomposition of boolean functions is considered as an important problem in the design of logic circuits and the area of research is considered as old as the area of digital circuit engineering. Boolean polynomials occur either directly or as a tool in the problem of decomposing boolean functions. The extensive use of integrated circuits that includes "modulo 2 adders", in electronics, draws attention towards the representation of boolean functions in the form of polynomials. Expressing boolean functions as boolean polynomials is considered an extensive method in boolean algebra. Boolean polynomials have a large number of applications in various fields including graph theory, law, medicine, operations research and spectroscopy [Bei93, CH08, Leo98, MS77, Rud04]. Polynomial methods have been employed extensively in circuit complexity and the boolean polynomials have variety of applications, *e.g.*, they used in Reed-Muller codes (error correcting codes). Besides a number of applications

including text mining, knowledge discovery, role engineering, sequential computation of boolean mappings also leads to decomposition of boolean matrices and directed graphs. Burckel et al. [Bur96, BGT09, BM04b] proposed *in situ* computation for boolean mappings and proved combinatorial results for the number of assignments involved. We illustrate these results in section 6.2 and 6.3 as related work to provide general background. Section 6.1 gives basic concept about boolean mappings whereas we propose strategy to develop sequence of assignments that computes boolean mappings (through construction of boolean polynomials) sequentially, in section 6.4.

6.1 Boolean Mappings

Boolean mappings describe how to determine boolean valued output based on some logical combinations from boolean inputs. These mappings can be represented in propositional logic, or as multivariate polynomials over $GF(2)$. These mappings are important in the theory of complexity as well as in the design of circuits/chips for digital computers. Besides, a number of applications exist in different other areas including artificial intelligence, propositional logic, electrical engineering, game theory, reliability theory and combinatorics. The properties of boolean mappings play a crucial role in cryptography, particularly in the design of symmetric key algorithms (a class of algorithms for cryptography that use boolean function keys for both decryption and encryption), *e.g.*, two fish, Serpent, Blowfish, CAST5, RC4, TDES, and IDEA.

Definition 6. *A boolean mapping 'f' is defined as*

$$f : \{0,\, 1\}^n \longrightarrow \{0,\, 1\}$$

The set $\{0,\, 1\}^n$*, of all n-tuples* $(x_1, \cdots, x_n)$*, where each* x_i *is either* 0 *or* 1*, is the domain for 'f'. There are* 2^{2^n}*, n-ary functions for every n.*

Example 19. *Let* $E(y_1,\ y_2,\ y_3)$ *be the boolean mapping from* $\{0,\, 1\}^3$ *to* $\{0,\, 1\}$ *such that*

$$\begin{aligned} y_1 &= x_1 + x_2 * x_3 \\ y_2 &= x_1 + x_2 + x_1 * x_3 + x_1 * x_2 + x_2 * x_3 \\ y_3 &= x_3 + x_1 * x_2 + x_2 * x_3 \end{aligned}$$

Components $(y_1,\ y_2,\ y_3)$ *of the mapping E are expressed in the table 6.1 that helps in verifying that the mapping E is bijective.*

Definition 7. *An in situ program that computes boolean mappings of the form*

$$E : \{0,\, 1\}^n \longrightarrow \{0,\, 1\}^n$$

consists in a sequence of assignments of one bit component defined as below

$$x_j := f_j\,(x_1, \cdots, x_n)$$

where $f_j : \{0,1\}^n \longrightarrow \{0,1\}$ *is a linear mapping and j is the index for input variables.*

x_1	x_2	x_3	y_1	y_2	y_3
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	1	0	0
1	0	0	1	1	0
1	0	1	1	0	1
1	1	0	1	1	1
1	1	1	0	1	1

Table 6.1: Boolean mapping

6.2 Computing Bijective Boolean Mappings

Burckel et al. [BGT09] proved that a bijective mapping on the set S^n is computed by an *in situ* program involving $2n-1$ number of assignments provided that n is the number of variables in the set of input variables. In this section, we illustrate this result for the case $S = \{0, 1\}$ in detail to provide a foundation for further work.

Theorem 7. *A bijective mapping E defined over $\{0, 1\}^n$ can be computed by an in situ program of the form.*

$$f_n, f_{n-1}, \cdots, f_3, f_2, f_1, g_2, g_3, \cdots, g_{n-1}, g_n$$

with $2n-1$ number of assignments involved in the program.

Proof. For $n = 1$, the *in situ* program takes the form $x_1 := f_1(x_1)$. To prove that the statement is true for $n > 1$, bipartite graph is used. Suppose that $G = (X, Y, A)$ be the bipartite multi-edges graph defined by $X = Y = \{0, 1\}^{n-1}$, and $(x, y) \in X \times Y$ is in A with the label $x_n y_n$, for $x_n y_n \in \{0, 1\}$ if and only if $E(x, x_n) = (y, y_n)$.
The degree of vertices of graph G will be exactly 2 due to the bijection E and graph G will be the union of disjoint even cycles due to the reason that it is 2-color-able regular bipartite graph. Therefore, by definition, graph G is 2-color-able [Hav07]. This is a particular case of a general result by König on regular graphs [Kön16], from which this proof can be generalized to any mapping on a finite set. Now, color the edges of G with elements of $\{0, 1\}$ and define two mappings E^0, E^1 on $\{0, 1\}^{n-1}$ and two mappings f_n, g_n from $\{0, 1\}^n$ to $\{0, 1\}$ as follows.
For each color $i \in \{0, 1\}$ and every edge (x, y) with color i and labeled $x_n y_n$, define

$$\begin{aligned} E^i(x) &= y \\ f_n(x, x_n) &= i \\ g_n(y, i) &= y_n \end{aligned}$$

By construction, any mapping E^i is bijective on $\{0,1\}^{n-1}$. Then under induction hypothesis, each E^i admits an *in situ* program of the form:

$$f^i_{n-1}, \cdots, f^i_2,\ f^i_1,\ g^i_2, \cdots, g^i_{n-1}$$

Define, for every $i \in \{0,\ 1\}$ and $x \in \{0,\ 1\}^{n-1}$.

$$f_j(x,i) = f^i_j(x)\ for\ j = n-1, \cdots, 1$$

$$g_j(x,i) = g^i_j(x)\ for\ j = 2, \cdots, n-1$$

In other words,

$$f_j(x,\ x_n) = x_n.f^1_j(x) + (1+x_n).f^0_j(x)\ for\ j = n-1, \cdots, 1$$

$$g_j(x,\ x_n) = x_n.g^1_j(x) + (1+x_n).g^0_j(x)\ for\ j = n-1, \cdots, 1$$

By construction, for the bijective boolean mapping E, we obtain an *in situ* program of length

$$2(n-1) - 1 + 2 = 2n - 1$$

as given below

$$\begin{aligned}
x_n &:= f_n(x_1, \cdots, x_n) \\
x_{n-1} &:= f_{n-1}(x_1, \cdots, x_n) = f^i_{n-1}(x_1, \cdots, x_{n-1}) \\
x_{n-2} &:= f_{n-2}(x_1, \cdots, x_n) = f^i_{n-2}(x_1, \cdots, x_{n-1}) \\
&\vdots \qquad \vdots \qquad \vdots \qquad \vdots \\
x_2 &:= f_2(x_1, \cdots, x_n) = f^i_2(x_1, \cdots, x_{n-1}) \\
x_1 &:= f_1(x_1, \cdots, x_n) = f^i_1(x_1, \cdots, x_{n-1}) \\
x_2 &:= g_2(x_1, \cdots, x_n) = g^i_2(x_1, \cdots, x_{n-1}) \\
&\vdots \qquad \vdots \qquad \vdots \qquad \vdots \\
x_{n-2} &:= g_{n-2}(x_1, \cdots, x_n) = g^i_{n-2}(x_1, \cdots, x_{n-1}) \\
x_{n-1} &:= g_{n-1}(x_1, \cdots, x_n) = g^i_{n-1}(x_1, \cdots, x_{n-1}) \\
x_n &:= g_n(x_1, \cdots, x_n) = g_n(y_1, \cdots, y_{n-1}, i)
\end{aligned}$$

In other words at the first step, the component x_n equals a color i, Then

$$E^i(x_1, \cdots, x_{n-1})$$

is computed by induction in $2(n-1)-1$ steps. At the step before last, we have

$$(x_1, \cdots, x_{n-1}) = (y_1, \cdots, y_{n-1})$$

At the last step we have $x_n = y_n$. □

Next, we explain Theorem 7 by giving two different examples.

6.2.1 Explanation

This section explains theorem 7 by examples.

Example 20. *Suppose that E_1 is a bijective boolean mapping defined over $\{0, 1\}^3$ and is represented by the truth table 6.2. Mapping E_1 can be computed*

x_1	x_2	x_3		x_1	x_2	x_3
0	0	0		0	0	0
0	0	1		1	1	1
0	1	0		0	0	1
0	1	1	$\xrightarrow{E_1}$	0	1	1
1	0	0		1	0	1
1	0	1		0	1	0
1	1	0		1	0	0
1	1	1		1	1	0

Table 6.2: Table for mapping E_1

by performing a sequence of operations, as given below.

$$x_1 := 1 + x_2 + x_1x_3 \qquad (f_1')$$
$$x_2 := 1 + x_1 + x_2 + x_1x_3 + x_2x_3 + x_1x_2 \qquad (f_2')$$
$$x_3 := x_1 + x_2 + x_3 + x_2x_3 \qquad (f_3')$$
$$x_2 := x_1 + x_2 + x_1x_3 \qquad (g_2')$$
$$x_1 := 1 + x_1 + x_2 + x_3 + x_2x_3 \qquad (g_1')$$

Sequence of operations consists in two sub-sequences. After performing the first three operations f_1', f_2', f_3', the mapping E_1 transforms into another boolean mapping that is still bijective then we perform the second sequence of operations g_2', g_1', and compute the mapping E_1. Table 6.3 explains, how the given mapping go through transformation process under the sequence of operations. The Intermediate truth table presents the given mapping after performing the

x_1	x_2	x_3		x_1	x_2	x_3		x_1	x_2	x_3
0	0	0		1	1	0		0	0	0
0	0	1		1	1	1		1	1	1
0	1	0		0	0	1		0	0	1
0	1	1	$\xrightarrow{f_1', f_2', f_3'}$	0	1	1	$\xrightarrow{g_2', g_1'}$	0	1	1
1	0	0		1	0	1		1	0	1
1	0	1		0	1	0		0	1	0
1	1	0		0	0	0		1	0	0
1	1	1		1	0	0		1	1	0

Table 6.3: Transformation process

sequence of first three operations. The final truth table shows the transformation obtained after performing the sequence of next two operations on the intermediate truth table.

Next, we describe a second example to explain all steps involve in the computation of boolean bijective mappings.

Example 21. *Let E_2 be a bijective boolean mapping over $\{0, 1\}^3$ as described in the Table 6.4. Given bijective mapping E_2 can be computed by performing*

x_1	x_2	x_3		x_1	x_2	x_3
0	0	0		1	1	1
0	0	1		0	1	1
0	1	0		1	1	0
0	1	1	$\xrightarrow{E_2}$	0	0	1
1	0	0		0	0	0
1	0	1		1	0	1
1	1	0		1	0	0
1	1	1		0	1	0

Table 6.4: Bijective boolean mapping (example)

the following sequence of operations.

$$x_1 := 1 + x_1 + x_2 + x_3 + x_2x_3 \qquad (f_1')$$
$$x_2 := x_1 + x_2 + x_1x_3 \qquad (f_2')$$
$$x_3 := x_3 + x_1x_2 \qquad (f_3')$$
$$x_2 := x_2 + x_3 + x_1x_3 \qquad (g_2')$$
$$x_1 := x_1 + x_2 + x_2x_3 \qquad (g_1')$$

We illustrate step by step the transformation of bijective mapping E_2 by applying the above sequence of operations.

x_1	x_2	x_3		x_1	x_2	x_3		x_1	x_2	x_3
0	0	0		1	0	0		1	1	0
0	0	1		0	0	1		0	0	1
0	1	0		0	1	0		0	1	0
0	1	1	$\xrightarrow{f_1'}$	0	1	1	$\xrightarrow{f_2'}$	0	1	1
1	0	0		0	0	0		0	0	0
1	0	1		1	0	1		1	0	1
1	1	0		1	1	0		1	0	0
1	1	1		1	1	1		1	1	1

Table 6.5: First Two Operations

Table 6.5 shows the transformation of given mapping E_2 after performing the first two operations f_1' and f_2'. The intermediate transformation is the transformation obtained after performing the very first operation. Transformation of given mapping E_2 under last three operations of the sequence of assignments is shown in the Table 6.6.

	x_1	x_2	x_3		x_1	x_2	x_3		x_1	x_2	x_3
	1	1	1		1	1	1		1	1	1
	0	0	1		0	1	1		0	1	1
	0	1	0		0	1	0		1	1	0
$\xrightarrow{f_3'}$	0	1	1	$\xrightarrow{g_2'}$	0	0	1	$\xrightarrow{g_1'}$	0	0	1
	0	0	0		0	0	0		0	0	0
	1	0	1		1	0	1		1	0	1
	1	0	0		1	0	0		1	0	0
	1	1	0		1	1	0		0	1	0

Table 6.6: Next Three Operations

It shows that total number of assignments involve in computing mapping E_2 is 5 that verifies the result proved in theorem 7.

Next, we describe an important property of the assignments involve in *"In Situ Design of Computation (IDC)"* of bijective boolean mappings.

6.2.2 Linearity Property

Burckel et al. [BGT09] proposed a linearity property of the assignments involve in *"In Situ Design of Computation (IDC)"* for bijective boolean mappings. We illustrate this property in detail as follows.

Lemma 2. *Every assignment $x_i := f_i(x_1, \cdots, x_n)$ performed in an in situ program to compute a boolean bijective mapping must be linear in x_i, i.e.*

$$f_i(x_1, \cdots, x_n) = x_i + h(x_1, \cdots, x_{i-1}, x_{i+1}, \cdots, x_n)$$

Proof. For $x_1 \in \{0, 1\}$, there exist two possible cases for the given mapping

$$f_i\,(x_1, x_2, x_3, \cdots, x_n)$$

i.e., the given mapping can hold one of the following forms

$$f_i\,(0, x_2, x_3 \cdots, x_n)$$

or

$$f_i\,(1, x_2, x_3 \cdots, x_n)$$

For each of these cases, there exists two possibilities for the output value, these possibilities are as given below.

$$f_i\,(0,\, x_2,\, x_3, \cdots, x_n) = 0 \quad \text{(a)}$$
$$f_i\,(0,\, x_2,\, x_3, \cdots, x_n) = 1 \quad \text{(b)}$$
$$f_i\,(1,\, x_2,\, x_3, \cdots, x_n) = 0 \quad \text{(c)}$$
$$f_i\,(1,\, x_2,\, x_3, \cdots, x_n) = 1 \quad \text{(d)}$$

In the computation of bijection, exactly, one of the equations from the pair (a, b) and one of the equations from the pair (c, d) will hold. Since the pairs of equations (a, c) and (b, d) cannot hold simultaneously due to the bijection, therefore, the only two possible pairs of equations (a, d) and (b, c) will hold. But the pair (a, d) will define the mapping

$$f_i\,(x_1,\, x_2,\, x_3, \cdots, x_n) = x_1$$

and the pair (b, c) defines

$$f_i\,(x_1,\, x_2,\, x_3, \cdots, x_n) = x_1 + 1$$

Therefore for all cases

$$f_i(x_1, \cdots, x_n) = x_1 + h(x_2, \cdots, x_n)$$

where

$$h(x_2, \cdots, x_n) = \begin{cases} 0 & \text{for the pair (a, d)} \\ & \text{and} \\ 1 & \text{for the pair (b, c)} \end{cases}$$

Hence for the *ith* component, we have

$$f_i(x_1, \cdots, x_n) = x_i + h(x_1, \cdots, x_{i-1},\, x_{i+1}, \cdots, x_n)$$

□

In the next section, we discuss the *"In Situ Design of Computation (IDC)"* for the case of general boolean mappings.

6.3 Computing General Boolean Mappings

The results included in this section have been proposed by Burckel et al. (see for instance [BGT09] and related work). We are interested to illustrate these results (concerning general boolean mappings over $\{0,\, 1\}^n$) in detail so that a foundation could be provided to proceed further. In general boolean mappings, it is possible that two different vectors may have the same image that is not the case for bijective mappings. We start with the following definition.

Definition 8. *A mapping 'f' on the set $\{0, \ldots, M\}$, where, $M > 0$, is called a step mapping if*

$$0 \leq f(y) - f(x) \leq y - x$$

such that

$$0 \leq x \leq y \leq M$$

In particular one can have the following

$$f(x) \leq f(x+1) \leq f(x) + 1$$

Next, we discuss simple mappings.

6.3.1 Simple Mappings

Simple mapping can be defined by making use of definition 8 and the mapping

$$\phi : \{0,\, 1\}^n \longrightarrow \{2^n\}$$

defined as given below

$$\phi(x_1,\, x_2,\, x_3, \ldots, x_n) = 2^{n-1}.x_n + \ldots + 2^2.x_3 + 2.x_2 + x_1,\ \forall\ n > 0$$

Definition 9. *A mapping E on $\{0,\, 1\}^n$ is called a simple mapping if*

$$E(x) = y \Leftrightarrow E'(\phi(x)) = \phi(y)$$

is a step mapping, where, E' is a mapping on $[2^n]$.

Proposition 3. *A simple mapping E on $\{0,\, 1\}^n$, $\forall\ n > 0$, can be computed by an in situ program of the form*

$$p_1,\, p_2, \ldots, p_n$$

with a length of 'n' assignments.
In brief, for $E(x_1,\, x_2,\, x_3, \ldots, x_n) = y_1,\, y_2,\, y_3, \ldots, y_n$ and for each $i = 1,\, 2, \ldots n$

$$p_i(y_1, \ldots, y_{i-1},\, x_i, \ldots, x_n) = y_i$$

Proof. Since the minimum number of assignments for n input variables is n. Therefore each function p_i must return its correct final value to each of its corresponding component x_i. This method for *in situ* computation of simple mapping is unique possible. However the correctness of this method is still remains to prove due to the fact that the mappings being computed are simple. □

Next, we describe the *"In Situ Design of Computation (IDC)"* for the general boolean mappings.

6.3.2 Decomposing general boolean mappings

In this section, we explain *"In Situ Design of Computation (IDC)"* of general boolean mappings. The idea consists in decomposing the mapping E of $\{0, 1\}^n$ in $F \circ P \circ G$ where F and G are bijective and P is simple.

Definition 10. *Suppose that E be a mapping on $\{0, 1\}^n$. A triple (F, P, G) on $\{0, 1\}^n$, with $n > 0$, of mappings is called a decomposition of the mapping 'E' if*

$$E = F \circ P \circ G$$

where, F and G are bijective mappings and P is a simple mapping.

Procedure

Suppose that E be the mapping on $\{0, 1\}^n$, the decomposition procedure for the mapping E consists in the following steps.

Step 1. In the first step, vectors with the same images are grouped via a bijective mapping G that gives intermediary consecutive images in lexicographical order to vectors with same final images. This will actually maps the sets $E^{-1}(x)$ for $x \in \{0, 1\}^n$ onto consecutive intervals of $[2^n]$ via $\phi \circ G$.

Step 2. The second step consists in the identification of vectors with the same final image via a simple mapping P such that $\phi \circ P \circ \phi^{-1}$ maps consecutive intervals of $[2^n]$ on consecutive values of $[2^n]$.

Step 3. The third step consists in the attribution of the correct final image value to each vector obtained by $P \circ G$ via another bijective mapping F, that is $F(P(G(x))) = E(x)$. Then F is completed arbitrarily to a bijective mapping on $\{0, 1\}^n$.

Next, we explain the result regarding an upper bound of the program that compute general boolean mappings.

Corollary 1 (see [BGT09] for more general case)**.** *Every mapping E on $\{0, 1\}^n$ can be computed by an in situ program of the form*

$$f_1, \ldots, f_n, g_{n-1}, \ldots, g_1, p_2, \ldots, p_{n-1}, f'_n, \ldots, f'_1, g'_2, \ldots, g'_n$$

such that the number of assignments (involve in the program) is $5n - 4$.

Proof. The procedure (6.3.2) of decomposition (given above) is used in constructing the proof. Suppose that $E = F \circ P \circ G$, *i.e.*, the triple (F, P, G) be the decomposition of E, with F and G are bijective mappings and P is a simple mapping. Both of the mappings F and G can be computed by a sequence of $2n - 1$ number of assignments (see Theorem 7). Similarly, the simple mapping

P can be computed by a sequence of n assignments as shown in Proposition 3. Combining these number of assignments, the mapping E, can be computed by a sequence of $5n-2$ number of assignments, of the form as given below

$$f_1, \ldots, f_n,\ g_{n-1}, \ldots, g_1,\ p_2, \ldots, p_{n-1},\ f'_n, \ldots, f'_1,\ g'_2, \ldots, g'_n$$

Two more assignments can be reduced by selecting a sequence of $2n-1$ assignments in such a way that it begins with the first variable for the mapping G and with the last variable for the mapping F. Moreover, any two successive assignments of the same component can be combined into a single assignment. Therefore, g_1, p_1 can be replaced by a single assignment g_1, and two assignments p_n, f'_n can be replaced by assignment f'_n.
Finally, we get a sequence of $5n-4$ assignments. □

Definition 11. *Let 'v' be the mapping defined by* $v : [2^n] \longrightarrow \mathbb{N}$*, then 'v' is called valuation if it satisfies the following equality*

$$v(0) + \ldots + v(2^n - 1) = 2^n$$

The value $l \in [2^n]$ *is denoted by* $v(l)$*, and* $A \subseteq [2^n]$ *by extension*

$$v(A) = \sum_{l \in A} v(l)$$

A valuation v is called boolean compatible if

$$\sum_{j2^i \leq l < (j+1)2^i} v(l) = 0 \ mod\ 2^i$$

such that

$$0 \leq i \leq n,\ and,\ 0 \leq j < 2^{n-i}$$

Definition 12. *Let v be the valuation, a mapping* $P_v : \{0,1\}^n \longrightarrow \{0,1\}^n$ *is called projection of v such that for l from* 0 *to* $2^n - 1$*,* $v(l)$ *consecutive elements of* $\{0, 1\}^n$ *are mapped onto* $\phi^{-1}(l)$*, beginning with* $(0, \ldots, 0) \in \{0, 1\}^n$ *for the first l with* $v(l) \neq 0$*.* P_v *is a simple mapping.*

Lemma 3. *Suppose that 'v' be a boolean-compatible valuation. Then for some* $k,\ k' \in [2^{n-i}]$*,*

$$P_v^{-1}(I_{i,j}) = \bigcup_{k \leq l \leq k'} I_{i,l}$$

where

$$I_{i,\,j} = \phi^{-1}([j2^i,\ (j+1)2^i - 1])$$

with

$$0 \leq i \leq n,\ and,\ j \in [2^{n-i}]$$

Proof. Suppose that the interval of $[2^n]$ is the converse image of an interval of $\{0, 1\}^n$ mapped by the mapping ϕ. Observe that, the converse image of an interval $\{0, 1\}^n$ is also an interval of $\{0, 1\}^n$ mapped under the mapping P_v (due to the reason that P_v is a simple mapping). Therefore, by definition of P_v, it gives

$$|P_v^{-1}(I_{i,j})| = \sum_{j2^i \leq l < (j+1)2^i} v(l)$$

For all $l \in \phi(I, i, j)$, if $v(l) = 0$ then $P_v^{-1}(I_{i,j})$ may be empty and it is given that v is a boolean compatible. Therefore, it gives

$$\sum_{j2^i \leq l < (j+1)2^i} v(l) = 0 \ (mod\ 2^i)$$

Thus

$$|P_v^{-1}(I_{i,j})| = 0 \ (mod\ 2^i)$$

By induction on j and for a fixed i, the result can be proved. Thus, if $j = 0$ then

$$|P_v^{-1}(I_{i,0})| = k.2^i, \text{ for some, } k \in [2^{n-i}]$$

Now, suppose that $P_v^{-1}(I_{i,0})$ is not empty, then by using definition of the projection P_v, it is an interval of $\{0, 1\}^n$ containing $(0, \ldots, 0)$. The length of this interval is $k.2^i$ which is the multiple of the factor 2^i, so, it is of the form as given below

$$\bigcup_{0 \leq l \leq k} I_{i,l}$$

Further, if the property is true for all l with $0 \leq l < j$, then

$$P_v^{-1}(\bigcup_{0 \leq l < j} I_{i,l}) = \bigcup_{0 \leq l < j'} I_{i,l}$$

Since

$$|P_v^{-1} I_{i,j}| = k.2^i, \text{ for some } k \in [2^{n-i}]$$

we must have

$$P_v^{-1}(\bigcup_{0 \leq l \leq j} I_{i,l}) = \bigcup_{0 \leq l \leq j'+k} I_{i,l}$$

Hence

$$P_v^{-1}(I_{i,j}) = \bigcup_{j' < l \leq j'+k'} I_{i,l}$$

□

Lemma 4. *Suppose that v be a boolean compatible valuation and $1 \leq i \leq n$. For $a, b \in \{0, 1\}^n$ and $l \geq i$, if $a_l = b_l$, then*

$$P_v(a)_l = P_v(b)_l \ , \ \forall \ l \geq i$$

Proof. Suppose that $a_l = b_l$, for $l \geq i$. Also, suppose that $c \in S^n$, where, $S = \{0, 1\}$ is given as under

$$c_n = a_n = b_n, \ldots, c_i = a_i = b_i, \; c_{k-1} = 0, \ldots, c_1 = 0$$

It gives

$$\phi(c) = 0 \; (mod \, 2^{i-1})$$

Thus for $j \in [2^{n-i+1}]$

$$\phi(c) = j.2^{i-1}$$

Moreover, $\phi(a)$ and $\phi(b)$ are in the same interval $[j.2^{i-1}, \; (j+1).2^{i-1} - 1]$ whose elements have same components for $l \geq i$. It gives that a and b belong to $I_{i-1,j}$. Now, using Lemma 3, the inverse images of intervals of the form $I_{i-1,k}$ by projection P_v are unions of such consecutive intervals. Hence the image of an interval $I_{i-1,j}$ by projection P_v is an interval contained in an interval $I_{i-1,k}$ for some $k \in [2_{n-i+1}]$.
Hence for $l \geq i$, $P_v(a)$ and $P_v(b)$ have the same components. □

Proposition 4 ([BGT09])**.** *Let E be a bijective mapping on $\{0, 1\}^n$ and is computable by an in situ program of the form $f_1, \ldots, f_n$. Then the mapping $E \circ P_v$ (where v is a boolean compatible valuation) is computed by an in situ program of the form $p_1, p_2, \ldots, p_n$. That is for $E(x_1, x_2, \ldots, x_n) = (y_1, \ldots, y_n)$ and for each $i = 1, 2, \ldots, n$:*

$$p_i(y_1, \ldots, y_{i-1}, x_i, \ldots, x_n) = y_i$$

Proof. To prove the assertion, the same argument is used as for the computation of simple mappings, *i.e.*, due to the fact that the number of assignments is minimal, necessarily each function p_i must gives its correct final value to each component x_i. But the correctness of the method is still to be proved.
On contrary basis, suppose that, at some step i, two different vectors x, x' become the same vector, say z, whereas their final images $y = E \circ P_v(x)$ and $y' = E \circ P_v(x')$ are different. Because, now, they become the same vector, one has

$$y_{i-1} = y'_{i-1} = z_{i-1}, \ldots, y_1 = y'_1 = z_1$$

On the other side, one has

$$x_n = x'_n = z_n, \ldots, x_i = x'_i = z_i$$

which implies, by Lemma 4

$$P_v(x)_l = P_v(x')_l = z'_l \, , \; \forall \, l \geq i$$

Now, suppose that

$$P_v(x) \neq P_v(x')$$

Since E is bijective, therefore, the mapping E' programmed by the sequence $f_1, \ldots, f_{i-1}$ is also bijective. But since E is programmed by $f_1, \ldots, f_n$, where, each component is modified at most once, it gives

$$E'(P_v(x)) = E'(P_v(x')) = (z'_n, \ldots, z'_i, z'_{i-1}, \ldots, z_1)$$

A contradiction. Hence $P_v(x) = P_v(x')$ and $y = y'$ □

Definition 13. *A boolean compatible decomposition of E defined over $\{0,1\}^n$ for $n > 0$ is a triple of mappings (F, P, G) on $\{0, 1\}^n$, such that $E = F \circ P \circ G$ with F and G are bijective mappings and P is the projection P_v of a boolean compatible valuation $v = v' \circ \sigma$, where $v'(l)$ is the number of elements of which image by E is $\phi(l)$.*

The construction of boolean compatible decomposition of a mapping is explained by the following Algorithm.

Algorithm 4: Boolean Compatible Decomposition

Algorithm: How to build a boolean compatible decomposition
Define a valuation v by $v(l) = |E^{-1}(\phi(l))|$;
Define a permutation σ such that $v \circ \sigma$ is boolean-compatible;
Define a bijection G of $\{0, 1\}^n$ compatible with $v \circ \sigma$;
begin
  Set i=0;
  for *l from 0* **to** $2^n - 1$ **do**
    for *j from 1* **to** $v \circ \sigma(l)$ **do**
      for *every* $x \in \{0, 1\}^n$ *with* $E(x) = \phi(l)$ **do**
        G(x):=$\phi^{-1}(i)$;
        i:=i+1;
      end
    end
  end
  Define $P = P_v$;
  Define a bijection F of $\{0, 1\}^n$ such that $E = F \circ P \circ G$;
  foreach *element* $x \in \{0, 1\}^n$ **do** F(P(G(x))):=E(x);
end
Complete the definition of F, keeping F bijective.

Theorem 8 (see [BGT09] for more detail)**.** *Every mapping E on $\{0, 1\}^n$ is computed by an in situ program of the form as given below*

$$f''_1, \ldots, f''_n, g''_{n-1}, \ldots, g''_2, f_1, \ldots, f_n, g'_{n-1}, \ldots, g'_1$$

such that the number of assignments involve is $4n - 3$.

Proof. Suppose that the triple (F, P, G) be boolean-compatible decomposition (by definition 13) of the mapping E. Now, since, F and G are bijective mappings, therefore, by Theorem 7, it requires $2n-1$ assignments to compute mappings F and G. Let

$$f'_1, \ldots, f'_n, g'_{n-1}, \ldots, g'_1$$

be the sequence of $2n-1$ assignments that compute mapping F and let

$$f''_1, \ldots, f''_n, g''_{n-1}, \ldots, g''_1$$

be the sequence of $2n-1$ assignments that compute mapping G respectively. Now, suppose that F' be the mapping on $\{0, 1\}^n$ computed by the sequence

$$f'_1, \ldots, f'_n$$

Then by Proposition (4), the mapping $F' \circ P$ is computed by a sequence of n assignments of the form

$$f_1, \ldots, f_n$$

Thus, the given mapping E is computed by sequence of assignments as given below

$$f''_1, \ldots, f''_n, g''_{n-1}, \ldots, g''_1, f_1, \ldots, f_n, g'_{n-1}, \ldots, g'_1$$

such that the number of assignments is $4n-2$, and the indices are the indices of the concerned variables. Notice that the assignments g''_1, f_1 can be replaced by a single assignment g_1. It results that the required number of assignments is $4n-3$. □

Next, we introduce method to construct IDC by constructing polynomials over $GF(2)$ with respect to set of boolean mappings.

6.4 IDC for Polynomials over GF(2)

Investigating decomposition of functions with respect to a given system of functions is a traditional problem of the theory of boolean functions. Decomposing boolean function is an important problem that deals with answering theoretical questions of the theory of boolean functions as well as in technical applications [VP93]. Some discussions about the polynomial form of boolean functions and its applications are presented in [Fal99, Sch98]. In this section, in the context of lemma 2 and discussion in previous sections, we propose a strategy to develop sequence of assignments (via boolean polynomials) that computes boolean bijective mappings. We start in constructing polynomials over GF(2) with respect to set of boolean mappings

$$y_1, y_2, \ldots, y_{i-1}, y_i, y_{i+1}, \ldots, y_n$$

and a set of their corresponding inverse mappings

$$x_1,\, x_2, \ldots, x_{i-1},\, x_i,\, x_{i+1}, \ldots, x_n$$

linear both in y_i and x_i. Construction of such polynomials enables to develop 'IDC' for bijective boolean mappings. As preliminary, $GF(2)$ (Galois field of two elements) is a finite field of elements 0 and 1 with the properties of addition and multiplication as given in the Table 6.7.

+	0	1
0	0	1
1	1	0

and,

*	0	1
0	0	0
1	0	1

Table 6.7: Properties of $GF(2)$

Next, we describe strategy that construct polynomials over $GF(2)$ with respect to set of boolean mappings and termed as "A First Tool".

6.4.1 A First Tool

This strategy consists in constructing polynomials over GF(2) with respect to set of boolean mappings such that the polynomials are linear with respect to the *ith* (say) mapping and its corresponding inverse image. The strategy works inductively and yields 'IDC' to compute boolean bijective mappings. Let E be a bijective boolean mapping of dimension n such that

$$\text{Set of boolean mappings} \rightarrow \begin{cases} y_1 & := f_1\,(x_1,\, x_2,\, x_3, \cdots, x_n) \\ y_2 & := f_2\,(x_1,\, x_2,\, x_3, \cdots, x_n) \\ y_3 & := f_3\,(x_1,\, x_2,\, x_3, \cdots, x_n) \\ \vdots & \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \\ y_i & := f_i\,(x_1,\, x_2,\, x_3, \cdots, x_n) \\ \vdots & \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \\ y_{n-1} & := f_{n-1}\,(x_1,\, x_2,\, x_3, \cdots, x_n) \\ y_n & := f_n\,(x_1,\, x_2,\, x_3, \cdots, x_n) \end{cases} \tag{SBM}$$

We are interested to compute polynomial linear with respect to mapping y_i and its inverse mapping x_i. For this purpose we proceed as follows.

Constructing Inverse Mappings

As a first step, we compute inverse images for each of the boolean mappings

$$y_1,\, y_2, \ldots, y_{i-1},\, y_i,\, y_{i+1}, \ldots, y_n$$

Corresponding to the set of boolean mappings (SBM), we denote by (SIM), the set of inverse images, as given below

$$\text{Set of inverse images} \rightarrow \begin{cases} x_1 & := g_1(y_1, y_2, y_3, \cdots, y_n) \\ x_2 & := g_2(y_1, y_2, y_3, \cdots, y_n) \\ x_3 & := g_3(y_1, y_2, y_3, \cdots, y_n) \\ \vdots & \vdots \ \vdots \ \vdots \ \vdots \ \vdots \ \vdots \ \vdots \\ x_i & := g_i(y_1, y_2, y_3, \cdots, y_n) \\ \vdots & \vdots \ \vdots \ \vdots \ \vdots \ \vdots \ \vdots \ \vdots \\ x_{n-1} & := g_{n-1}(y_1, y_2, y_3, \cdots, y_n) \\ x_n & := g_n(y_1, y_2, y_3, \cdots, y_n) \end{cases} \quad \text{(SIM)}$$

Computing Possible Products

Compute all possible products of boolean mappings contained in the set (SBM), *i.e.*, compute possible products of permutations of

$$y_1, y_2, \ldots, y_{i-1}, y_i, y_{i+1}, \ldots, y_n$$

These products are of the form as given below.

$$\text{Possible products} \rightarrow \begin{cases} y_1y_2, y_1y_3, \cdots, y_1y_n \\ y_2y_3, y_2y_4, \cdots, y_2y_n \\ y_3y_4, y_3y_5, \cdots, y_3y_n \\ \vdots \ \vdots \ \vdots \ \vdots \ \vdots \ \vdots \\ y_1y_2y_3, y_1y_2y_4, \cdots, y_1y_2y_n \\ y_1y_3y_4, y_1y_3y_5, \cdots, y_1y_3y_n \\ \vdots \ \vdots \ \vdots \ \vdots \ \vdots \ \vdots \ \vdots \ \vdots \\ y_1y_2y_3y_4, y_1y_2y_3y_5, \cdots, y_1y_2y_3y_n \\ y_1y_2y_3y_4y_5, y_1y_2y_3y_4y_6, \cdots, y_1y_2y_3y_4y_n \\ \vdots \ \vdots \ \vdots \ \vdots \ \vdots \ \vdots \ \vdots \ \vdots \ \vdots \ \vdots \ \vdots \end{cases}$$

Computing Possible Products of Inverse Mappings

Compute all possible products for the set of mappings (SIM), *i.e.*, compute the possible products of permutations of

$$x_1, x_2, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_n$$

These products will be of the form as given below.

$$\text{Possible products of inverse mappings} \rightarrow \begin{cases} x_1x_2,\ x_1x_3, \cdots, x_1x_n \\ x_2x_3,\ x_2x_4, \cdots, x_2x_n \\ x_3x_4,\ x_3x_5, \cdots, x_3x_n \\ \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \\ x_1x_2x_3,\ x_1x_2x_4, \cdots, x_1x_2x_n \\ x_1x_3x_4,\ x_1x_3x_5, \cdots, x_1x_3x_n \\ \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \\ x_1x_2x_3x_4,\ x_1x_2x_3x_5, \cdots, x_1x_2x_3x_n \\ x_1x_2x_3x_4x_5, x_1x_2x_3x_4x_6, \cdots, x_1x_2x_3x_4x_n \\ \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \end{cases}$$

Require

For an *ith* mapping of the set (SBM) and the corresponding inverse mapping of the set (SIM), the following equality must be satisfied

$$y_i + h_i\left(y_1,\ y_2, \ldots, y_{i-1},\ y_{i+1}, \ldots, y_n\right) = x_i + \acute{h_i}\left(x_1,\ x_2, \ldots, x_{i-1},\ x_{i+1}, \ldots, x_n\right) \tag{6.1}$$

Establishing the System

To satisfy equality 6.1, establish a system of equations. For this purpose, introduce a set of coefficients

$$c_1,\ c_2, \ldots, c_{i-1},\ c_{i+1}, \ldots, c_n$$

and start in generating one side of the equality 6.1, for instance, for *ith* component, one of the possible expressions could be of the form

$$y_i + a_1y_2 + a_2y_3 + a_3y_2y_3 + \ldots \tag{6.2}$$

Then, compute values for the coefficients and evaluate the expression 6.2. Finally, one can get the required polynomial by substituting back the values of these coefficients and evaluating the corresponding expression.

A quick view of the procedure is given in the figure 6.1, where, 'SBM' denotes the set of boolean mappings (corresponding to bijective mapping) and 'SIM' denotes the set of inverse images corresponding to the set 'SBM'. Yellow bar shows the process of establishing and evaluating the system of equations. Different steps included in the process are shown by arrows.

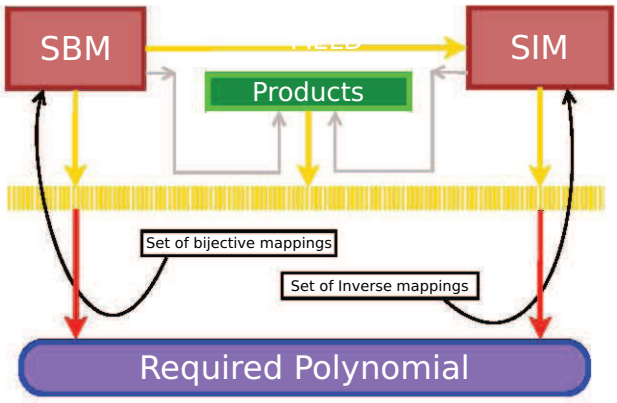


Figure 6.1: Interpretation of "A First Tool"

In this way, we develop basic step (as given below) that leads to construct IDC through polynomials over $GF(2)$.

$$\left.\begin{array}{l} y_1 + h_1\,(y_2,\, y_3, \ldots, y_{i-1},\, y_i,\, y_{i+1}, \ldots, y_n) := \\ x_1 + \acute{h}_1\,(x_2,\, x_3, \ldots, x_{i-1},\, x_i,\, x_{i+1}, \ldots, x_n) \end{array}\right\}$$

This polynomial (basic step) provides two assignments, the first assignment and the last assignment, that are involve in computing bijective boolean mappings.

6.4.2 Explanation and Construction

In this section, we explain method "A First Tool" by constructing the first required polynomial over GF(2). Let $E : (x_1, x_2, x_3) \rightarrow (y_1, y_2, y_3)$ be a bijective mapping (see table 6.8), such that

$$\left\{\begin{array}{l} y_1 := 1 + x_2 + x_1x_3 \\ y_2 := 1 + x_1 + x_2 + x_1x_2 + x_2x_3 + x_1x_3 \\ y_3 := x_1 + x_2 + x_3 + x_2x_3 \end{array}\right. \qquad \text{(SB)}$$

To compute polynomial linear both in y_1 as well as in x_1, we compute the possible products as given below:

$$\text{Possible products} \rightarrow \left\{\begin{array}{ll} y_1y_2 & := 1 + x_1 + x_2 + x_1x_2 \\ y_1y_3 & := x_1 + x_3 + x_1x_2 + x_2x_3 \\ y_2y_3 & := x_3 + x_1x_3 \\ y_1y_2y_3 & := x_3 + x_2x_3 + x_1x_3 + x_1x_2x_3 \end{array}\right.$$

Then, corresponding to the set of boolean mappings (SB), we compute a set of inverse mappings as shown below.

$$\text{Set of inverse mappings} \rightarrow \left\{\begin{array}{l} x_1 := 1 + y_1y_3 + y_2y_1 + y_3 \\ x_2 := 1 + y_1y_3 + y_2y_3 + y_2 \\ x_3 := y_1 + y_2 + y_1y_3 \end{array}\right. \qquad \text{(SI)}$$

x_1	x_2	x_3	y_1	y_2	y_3
0	0	0	1	1	0
0	0	1	1	1	1
0	1	0	0	0	1
0	1	1	0	1	1
1	0	0	1	0	1
1	0	1	0	1	0
1	1	0	0	0	0
1	1	1	1	0	0

Table 6.8: Bijective Mapping E

Then, we compute the possible products for the set of inverse mappings (SI), that are listed below

$$\text{Possible products of inverse mappings} \rightarrow \begin{cases} x_1x_2 & := 1 + y_2 + y_3 + y_2y_3 \\ x_1x_3 & := y_1 + y_2 + y_1y_3 + y_2y_3 \\ x_2x_3 & := y_1 + y_2y_3 + y_1y_3 + y_2y_1 \\ x_1x_2x_3 & := y_1 + y_1y_3 + y_2y_1 + y_1y_3y_2 \end{cases}$$

Now, we introduce three coefficients, α, β and γ. To construct the required polynomial, we need to make an expression of the form

$$y_1 + \alpha y_2 + \beta y_3 + \gamma(y_2 * y_3) \tag{6.3}$$

that gives

$$y_1 + \alpha y_2 + \beta y_3 + \gamma(y_2 * y_3) = \begin{cases} & (1 + x_2 + x_1x_3) \\ + & \alpha\,(1 + x_1 + x_2 + x_1x_2 + x_2x_3 + x_1x_3) \\ + & \beta\,(x_1 + x_2 + x_3 + x_2x_3) \\ + & \gamma\,(x_3 + x_1x_3) \end{cases}$$

Establish a system of equations by comparing the coefficients (in the above equality) of unwanted products, *e.g.*, x_1x_3 etc. Therefore

$$\begin{cases} 1 + \alpha + \gamma & = 0 \quad \text{(comparing coefficients of } x_1x_3\text{)} \\ \alpha & = 0 \quad \text{(comparing coefficients of } x_1x_2\text{)} \\ \alpha + \beta & = 1 \quad \text{(comparing coefficients of } x_1\text{)} \end{cases}$$

It gives

$$\alpha = 0, \;\; \beta = 1, \quad \text{and} \quad \gamma = 1$$

Substituting these values of α, β and γ in (6.3), we get the required polynomial

$$x_2x_3 + x_1 := 1 + y_1 + y_2y_3 + y_3 \tag{6.4}$$

that is linear in y_1 and x_1. This is the basic step that leads to design sequence of *"In Situ Design of Computation (IDC)"* for the boolean bijective mapping E.

Example 22. *Let* $E : (x_1, x_2, x_3) \rightarrow (y_1, y_2, y_3)$ *be boolean bijective mapping such that*

$$\begin{cases} y_1 := x_1 + x_2x_3 + x_1x_3 \\ y_2 := x_1 + x_2 + x_1x_2 + x_2x_3 \\ y_3 := x_1 + x_1x_2 + x_2x_3 + x_1x_3 + x_3 \end{cases} \quad \text{(Set of mappings)}$$

where, table 6.9 presents boolean values that construct boolean bijective mapping E*. Let* A *denotes polynomial over GF(2) linear both in* y_1 *as well as in* x_1*.*

x_1	x_2	x_3	y_1	y_2	y_3
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	1	0	0
1	0	0	1	1	1
1	0	1	0	1	1
1	1	0	1	1	0
1	1	1	1	0	1

Table 6.9: Bijective Mapping E

There are two such polynomials for the set of mappings y_1*,* y_2 *and* y_3*. One of them is written as follows:*

$$A := x_2x_3 + x_1 + x_2 + x_3 = y_1 + y_2 + y_3 \tag{6.5}$$

Polynomial (6.5) leads to generate 'IDC' for the mapping E *by performing the following steps:*

- *Construct first assignment by setting* A *equals* x_1 *as*

$$x_1 := x_2x_3 + x_1 + x_2 + x_3 \tag{6.6}$$

- *Construct two mappings* E_1 *and* E_2 *by letting* $A = 0$ *and* $A = 1$ *respectively.*

Table 6.10 presents mapping E_1 *and Table 6.11 presents mapping* E_2*. Mappings*

x_2	x_3	y_2	y_3
0	0	1	1
0	1	0	1
1	0	1	0
1	1	0	0

Table 6.10: Mapping E_1

presenting table 6.10 are defined as

$$y_2 = 1 + x_3$$

$$y_3 = 1 + x_2$$

- *Construct polynomial linear in x_2 and y_2 and is given below*

$$x_2 + x_3 = y_2 + y_3$$

Now, sequence of 'IDC' that computes mapping E_1 is

$$\begin{cases} x_2 := x_2 + x_3 \\ x_3 := 1 + x_2 + x_3 \\ x_2 := x_2 + x_3 \end{cases} \quad \text{(IDC for } E_1\text{)}$$

x_2	x_3	y_2	y_3
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	1

Table 6.11: Mapping E_2

Mappings presenting table 6.11 are defined as

$$y_2 = x_2 + x_3$$

$$y_3 = x_3$$

- *Construct polynomial linear in x_2 and y_2 and that is*

$$x_2 + x_3 = y_2$$

Now, sequence of 'IDC' that computes mapping E_2 is

$$\begin{cases} x_2 := x_2 + x_3 \\ x_3 := x_3 \\ x_2 := x_2 \end{cases} \quad \text{(IDC for } E_2\text{)}$$

- *Combine the sequence of IDC for E_1 and the sequence of IDC for E_2 as given below*

$$\begin{aligned} x_2 &:= (1 + x_1)(x_2 + x_3) + x_1(x_2 + x_3) \\ x_3 &:= (1 + x_1)(x_3) + x_1(1 + x_2 + x_3) \\ x_2 &:= (1 + x_1)(x_2) + x_1(x_2 + x_3) \end{aligned}$$

It gives three intermediate assignments as given below

$$\begin{aligned} x_2 &:= x_2 + x_3 \\ x_3 &:= x_1 + x_3 + x_1x_2 \\ x_2 &:= x_2 + x_1x_3 \end{aligned}$$

- *Construct the sequence of 'IDC' that computes mapping E as follows:*

$$\begin{cases} x_1 := x_1 + x_2 + x_3 + x_2x_3 \\ x_2 := x_2 + x_3 \\ x_3 := x_1 + x_3 + x_1x_2 \\ x_2 := x_2 + x_1x_3 \\ x_1 := x_1 + x_2 + x_3 \end{cases} \quad \text{(IDC for E)}$$

Conclusion of the part

The strategies for *in situ* computation of mappings provide a strong foundation to proceed further in this direction of research and to enhance the capabilities of sequential computations. Along with, a quick glance of applications in matrix decomposition, it motivates to implement the idea in circuit/chip design. A number of publications (see for instance [Pin99]) could be found for further motivations. The idea of *in situ* computation is a great effort in contributing to optimization (for example processor/compiler/memory) and could be extended further by finding real applications and implementations. But still there are limitations that need attention to be resolved and the strategies need some modifications to deal with such limitations.

To highlight these limitations, in chapter 3, we have explained the idea of *in situ* computation for mappings over fields and provided counter example that leads towards the modifications of the existing approach. An alternate approach (using Bézout's Identity) to generate sequence of assignments that performs *in situ* computation for mappings with two dimensions has been discussed but the idea needs its further extension for general case and it requires to investigate the possibility for the case of integers.

Similar limitations are observed in the case of rings and are highlighted by providing counter example in generating sequence of assignments (from the sequence of assignments that compute given mappings) to compute inverse mappings. Thus, current approach requires modifications to deal with such situations and could be applied possibly in different way.

In case of integers, (chapter 5), the combinatorial results and counter examples show that there are certain limitations for *in situ* strategies and it require improvements and modifications. For example, reducing the number of assignments (involved in computation), providing the possibility to generate sequence of assignments that could compute inverse mapping and extending the approach for general case. Moreover, in reducing number of assignments, we have to deal carefully with some situations, *e.g*, packing similar assignments

(involved in computation) may results in an assignment with a co-efficient to be multiplied.

A tool for generating sequence of assignments (chapter 6) for *in situ* computation of bijective boolean mappings through construction of polynomials over GF(2) is a basic attempt to develop the required strategy. But still, it remains to extend the idea for general case and to find some useful combinatorial results.

Part II

Decentralized Collaborative Editing

Chapter 7

Collaborative Editing System

Contents

7.1 Introduction 103
7.1.1 A Collaborative Editing Scenario 104
7.1.2 Pessimistic Concurrency Control 105
7.1.3 Optimistic Concurrency Control 106
7.1.4 Group Communication 106
7.1.5 Centralized collaborative editing systems 106
7.1.6 Decentralized collaborative editing systems . . . 108
7.2 DCE Model 111
7.2.1 Framework Description 113
7.2.2 Editing a Document 114
7.2.3 Modifying a Document 115
7.2.4 Mapping between elements and Identifiers . . . 115

7.1 Introduction

Collaborative Editing Systems (CES), provide environment to group of users, dispersed geographically, so that they can edit/modify the same data concurrently. For this purpose, collaborative editing tools have been designed that enable authorized users to edit/modify a shared document, to see who else is working over the same document and to know in real time, the changes that are being made by others. Wikis, on-line office suites and version control systems are the most popular collaborative editing tools [Cit07]. The shared documents are similar to wikis in the sense that a number of users can make modification by adding or deleting the content. Some of the collaborative editing tools facilitate users to communicate by instant messaging (like a chat

session) as an addition to collaborative editing. As an example, Wikipedia is edited by 7.5 millions of users and got 10 millions of articles in only six years [WUM09].

7.1.1 A Collaborative Editing Scenario

To understand, how the collaborative editing works, consider a collaborative editing scenario as presented in the Figure 7.1.

Three users are participating in editing a shared document contained an initial content ”behave”. The elapse of time is shown by vertical lines. An operation generated locally is presented by circle and is executed immediately. The propagation of local operations to other sites are presented by arrows. Suppose that user-1 inserts character “r“ at position 2 of the initial content,

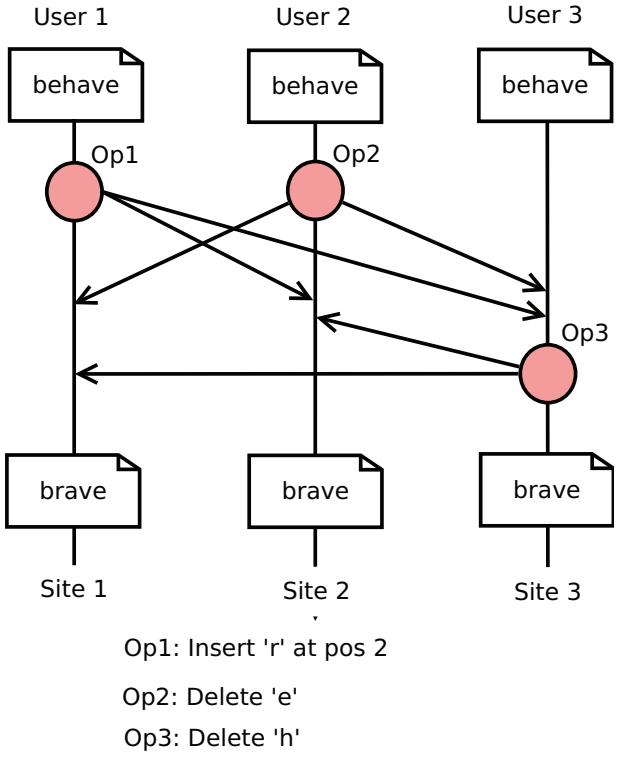


Figure 7.1: A collaborative editing scenario

and user-2 deletes character “e“ at position 2, simultaneously. Each replica is updated after the execution of operations “Op1“ and “Op2“ so that user-3 has replica modified to “brhave“. Now user-3 performs operation “Op3“ and deletes “h“ at position 3 of the modified copy of the initial content. Updating the document after executing operation ”Op3” results in containing the final content as ”brave”.

Another simple example of collaborative editing is shown in Figure 7.2 (next page), where INRIA research centres are joined with each other. Suppose users

at different centres participate in editing an INRIA Research Report. A well defined collaborative editing system (for example, INRIA Gforge) makes it possible that all users can write a common research report.

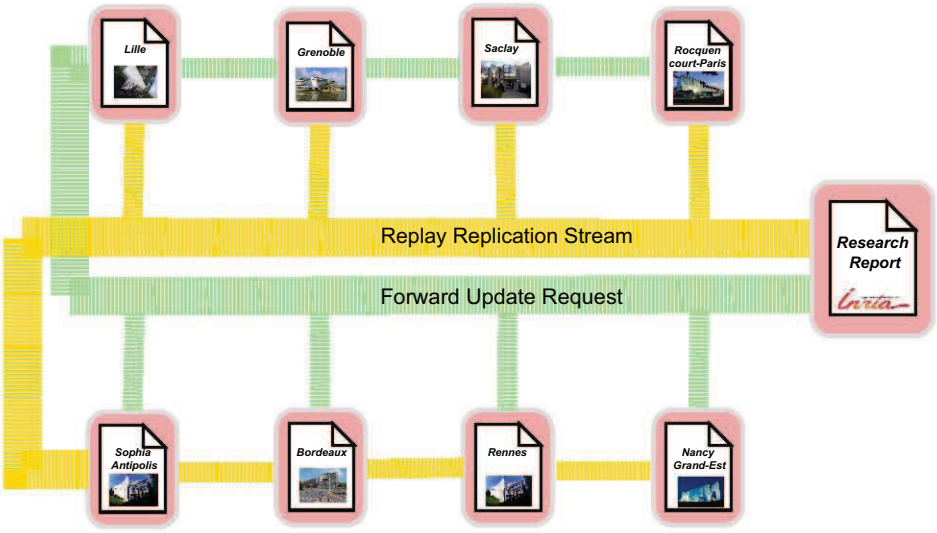


Figure 7.2: An example of collaborative editing

To make it possible that all participants in real time collaborative editing, edit and view the shared document, almost at the same time, some kind of synchronization strategies between the participants must be used. Traditionally, there exist two synchronization control approaches, in reaching this behavior [Pre07], they are pessimistic concurrency control (lock based) and optimistic concurrency control (lock free).

7.1.2 Pessimistic Concurrency Control

Pessimistic concurrency control (Lock based editing), is based on acquiring a lock before starting edition for a given object/document. When a lock is applied to facilitate one user, all other participants have to wait until the lock is released. Therefore, only one user can edit a given object/document at a time, that in some ways limits the parallel nature of collaborative editing. Briefly, it can be described in the following three steps.

- Acquire the lock on the object “O“, given that it is not already acquired.
- Perform some edits to the object.
- Release the lock on the object ”O”.

Another relevant issue with lock based editing is when a participant actively editing an object, it crashes or leaving the object locked forever.

7.1.3 Optimistic Concurrency Control

A fast response and high degree of concurrency is the need of Real-time Applications. Optimistic concurrency control has the properties of non-blocking and deadlock freedom, they are especially recommended to real-time transaction processing [HCL90, HSRT91, JS92]. Lock free editing, or optimistic concurrency control, allows participants to more freely modify an object in a more parallel nature. However, there are risks of conflicts that needs some kind of algorithms to resolve these conflicts automatically. Further, it may results in loss of data, when, for example, a user's edits are overwritten during the conflict resolving [CMR07, JBC05].

Collaborative editing between more than two participants require some kind of group communication. However, different collaboration strategies require different styles of group communication guarantees.

7.1.4 Group Communication

Group communication allows reliable messaging between senders and receivers comprise a network. In the basic form of group communication, two users communicate with each other while others wait for their turn. Some of the problems regarding group communication (for instance, reliability and ordering) are addressed using different techniques but still there exist some other problems, when multi users are participating simultaneously. More difficult problems occur in providing the reliability for transmission and ordering of packets etc [Å10].

Total Ordering

To maintain the consistency among the replicas, a total ordering on the updates must be provided. Total ordering within group communication guarantees that all broadcast messages will be received in the same order, by all recipients. Being able to totally order the events can be very useful in implementing distributed system. To see the importance of total ordering consider an example of two clients C1 and C2. Suppose these two clients send updates for a document "D" simultaneously, Suppose these updates are totally ordered, so the servers either S1 receive first the update from C1 and then the update from C2 or S2 receive first the update from C2 and then the update from C1. In both cases, the replicas will remain consistent, since every server applies updates in the same order. If these updates are not totally ordered, the servers can apply updates in different orders that will cause the consistency problem [Å10].

7.1.5 Centralized collaborative editing systems

In centralized systems, the messages have to go through a round trip and users cannot see their own actions immediately. Therefore, local commands (includ-

ing lock requests) are not of responsiveness. Existing collaborative editing systems such as CVS, RCS or WIKIs are centralized and unanimously adopt the client-server architecture. The server node holds a persistent copy of shared document. Each client node stores a copy of the shared document. A user at a client node updates the shared document through the local copy. All updates are synchronized to other users through the server node. Such systems cannot be adapted to peer-to-peer networks [OUMI06].

Below, we discuss some collaborative editing systems keeping the order of their restrictiveness on collaboration.

RCS

RCS (Revision Control System) is a version control system where a single or multiple users modify document through an explicit check-out step. In RCS, when user attempts to check in a new version whose modifications are based on a state version, editing conflicts occur. To detect these editing conflicts, RCS uses a locking mechanism. It notifies impacted users through diagnostic messages. Usually RCS is being used to handle source code in software development, however, it is also used to support wiki applications, *e.g.*, Twiki [twi10].

MediaWiki

In MediaWiki, users are able to edit different parts of a document without interference. If multi-user edit the same paragraph simultaneously, it causes editing conflicts. A user releases edits by clicking a button manually. MediaWiki uses diff3 [com10] to merge changes, automatically, that make by users but these changes should be made in different parts of the document, otherwise, impacted users are notified by diagnostic messages. MediaWiki is working as underlying engine for the largest on-line encyclopedia, Wikipedia [kao10b].

Google Docs

Google Docs serves as collaborative tool and it combines, features of Writely and Spreadsheets with a presentation program incorporating technology designed by Tonic Systems. Since, January 13, 2010, a data storage capacity of 1GB is introduced. Editing conflicts occur if more than one users simultaneously update the same sentence. Versions of Google Docs for the iPhone and Android include functionality for editing spreadsheets and viewing presentations, along with an interface designed specifically for the device. Updates between users are automatically synchronized within seconds and to merge changes from different users, it uses differential-synchronization algorithm [kao10a].

Google Wave

Google Wave allows users to edit a shared document anywhere and anytime that is why it considered as most liberal editing system. The system is able to reconciles conflicts of edition, in all situations including, when users edit an overlapping area simultaneously. Google Wave uses operational transformation (OT) [EG89], a non-blocking distributed concurrency control algorithm to guarantee the data consistency. It also preserve the properties of convergence and causality. The causality preservation follows Lamport's logical clock [Lam78], that needs all operations to be executed in their happened-before relationships.

The collaborative editing systems described above differ by the degree of restriction on collaboration and each of them uses different implementation techniques. For instance, RCS (Revision Control System) uses locking mechanism, whereas Google Wave uses operational transformation [EG89] to guarantee the data consistency.

7.1.6 Decentralized collaborative editing systems

Decentralized collaborative systems provide environment in which a number of users located at several sites are able to update common object (*e.g.*, text document) independently. Each participant in the network has a separate replica (*i.e.*, local copy) of the document and he/she modifies his/her replica. Editing a shared text or document co-operatively, is considered to be a well-studied example. When users make any modifications locally, the replicas diverge from one another. The operations performed on some sites propagate to other sites and are integrated or replayed there. Eventually, all sites execute every action of users.

Collaborative systems based on the operational transformation approach could be decentralized [EG89, SE98, WUM09]. The replicas might not converge if the users execute operations in different orders. For convergence, two basic approaches can be found in the literature. One of them is the serializing, *i.e.*, enforcing a total order of operations before execution [Lam78]. Second approach is operational transformation, *i.e.*, modifying the parameters of operations to make them run in different orders [SE98]. The later approach is considered to be complex and error-prone, as evidenced by the errors found in published algorithms [OUMI05].

Consistency Issue

To maintain the consistency in a scalable and decentralized manner is a challenging problem due to the replication and exchange of updates. Concurrency control techniques, such as (pessimistic/optimistic) locking and serialization, are no more considered to be effective because they may ensure consistency at the cost of responsiveness and loss of updates [EG89, LL04a, SJZ+98]. To illus-

trate this problem, we present a scenario in Figure 7.3. where two users user-1

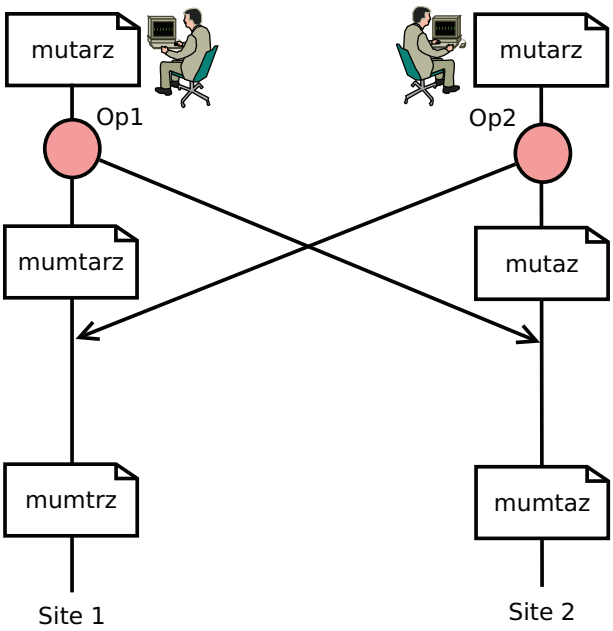


Figure 7.3: Wrong Integration

and user-2 work on a shared document represented by sequence of characters. These characters are labeled by a sequence of natural numbers. Suppose that, both users, initially, have a sequence “mutarz“ of characters. Let user-1 executes an operation “Op1“ and inserts a character “m“ at position 3, user-2 performs operation “Op2“ and deletes character “r“ at position 5. Now, Op1 is received at site 2 and after execution, it produces a sequence of characters “mumtaz“. But, at site 1, “Op2“ has not the same effect because it produces a different sequence “mumtrz“ as compared to the sequence produced at site 2. In spite of it, that all sites execute operations in the same order following the requirement of serialization protocol [EG89], the final results are not the same.

Operational Transformation

Operational transformation (OT) is basically invented for consistency maintenance and concurrency control in collaborative editing of plain text documents. In this approach, local operations are executed immediately after their generation, while remote operations must be transformed regarding concurrently executed operations. To address the consistency issue, an OT approach has been proposed in [EG89, Imi08]. Generally, it is an application-dependent transformation algorithm, called IT, that works in a way that for every possible pair of concurrent updates, the application programmer has to specify

how to integrate these updates regardless of reception order. To see the effect of IT on previous example, consider the Figure 7.4. To have the same final

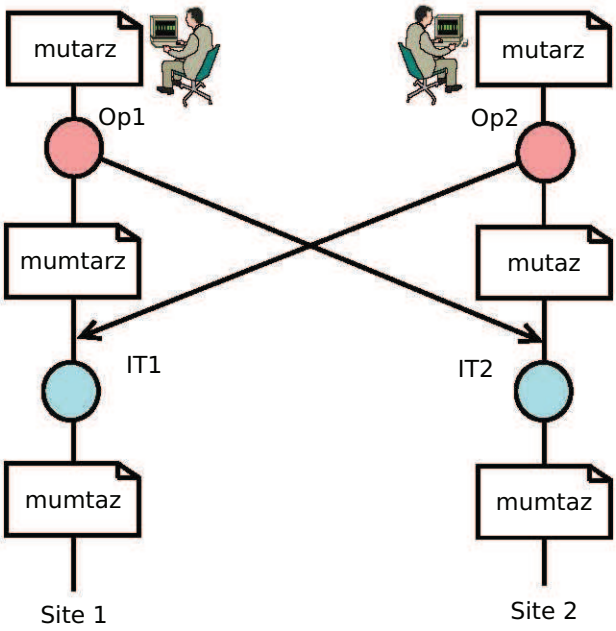


Figure 7.4: Correct Integration

result, on site 1 and site 2, operation ”Op2” has to be operated in a way such that IT1(Op1, Op2)=Del(6). The operation IT1 includes the effect of both operations ”Op1” and ”Op2”. The main task of OT is to assure consistency in a decentralized way without the requirement of any global order. OT facilitates users in a way so that they can modify shared document concurrently and exchange updates in any order. But still most of the OT algorithms does not guarantee consistency, because there exist some bugs. The proposed OT environments dealt with a fixed number of users in collaborative session. Moreover, scalability requirements are still needs attention from the OT research community [IMOR03].

We aim to design decentralized collaborative editing system that could be able to serve in a way better than the existing systems. In the next section, we propose a model and describe our idea in detail:

7.2 DCE Model

Our DCE (decentralized collaborative editing) model is based on precision control indexing method that we will discuss later in detail. Our idea is applicable to all kind of documents that bear sequential structure but, here, for the sake of simplicity, we illustrate it by taking a shared text document. To edit the shared document, an Insert or Remove (delete) operation is followed by an Update to synchronize the document replica (as shown in Figure 7.5). Insert operation, requires in collecting information of neighboring elements, and computing new identifier for an entry to be inserted. Therefore, a Collect action collects the in formation and Comp will compute new identifier. One of the advantages of the technique is that, for an Insert operation, it needs the informations only about two identifiers corresponding to characters neighboring the insertion position. For a Delete operation, it needs information exactly about the identifier corresponding to element (character, line, etc) to be deleted. This is important because it avoids communication overhead for moving data items between machines given that Read operations are frequent and may involve a large amount of data items. Since all document replicas are updated in a global serialization order both the convergence property and the causality preservation property (see section 7.2.1) are preserved. In this model, users update the shared document without any restriction. All editing conflicts are automatically reconciled. Our model can be treated as update-anywhere-anytime Model. Further, the model can be easily interpreted as a network $\aleph$ of n ($n \in \mathbb{N}$) users (sites or peers) that assigned a unique small positive real value $''\epsilon''$ (chosen from a set of such values $\{\epsilon_i,\ i = 1, 2, \cdots n\}$, generated under specific precision), to each user (site or peer) such that these values act as identifiers for users (sites or peers). In our approach, a shared document is mapped to an interval $I = [\,a,\, b\,]$, where, $0 \leq a < b$, for $a, b \in \mathbb{R}$ and unique identifiers are computed over the interval I such that these unique identifiers could be assigned to characters or lines (to be edited) in the shared document. Corresponding to the insertion or deletion of elements in the document, there exists insertion/deletion of identifiers inside the interval I. The idea of associating identifiers resembles with the idea of shared sequential buffer

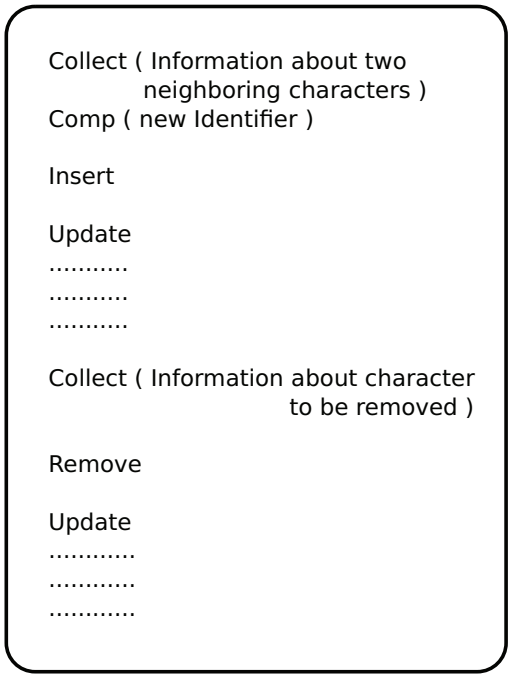


Figure 7.5: Model

where each buffer entry is identified by unique identifier [PMSL09]. In a shared document, an element may present a sequence of characters assigned a unique identifier with the following properties:

1. Each element is assigned an identifier.
2. Two elements have two different identifiers.
3. The identifier once assigned does not change during the whole lifetime of the document
4. There exists a total order on identifiers, $<$, consistent with the order of elements.
5. For two identifiers $ID1 < ID2$, new identifier $ID3$ respects the inequality $ID1 < ID3 < ID2$.

At the completion of process of insertion/deletion, the final set of unique-identifiers will be an ordered set. In fact, an insertion of a point partitions the interval I into further sub-intervals. Thus, an insertion of $m \in \mathbb{N}$ points in the interval I takes the form

$$a = a_0 < a_1 < a_2 < \ldots < a_{m-1} < a_m < a_{m+1} = b$$

such that

$$a_m = a_{m-1} + \frac{a_{m+1} - a_{m-1}}{2} - \epsilon$$

respecting rounding and computation conditions introduced to the Algorithm 8 (will be discussed in detail, later in chapter 8, section 8.1.3) used to create these identifiers. Consider the Figure 7.6, a sequence of characters is indexed

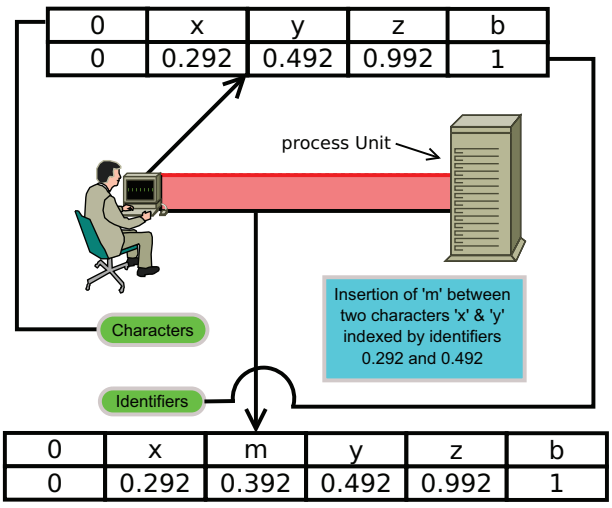


Figure 7.6: Inserting single character

by a sequence of identifiers. The initial and final characters are marked with "0" and "b", whereas the initial and final points for the identifiers are marked with "0" and "1". A user is intended to insert a character "m" between two neighboring characters "x" and "y" indexed by two identifiers 0.292 and 0.492. These identifiers are calculated using precision control indexing method (will be discussed in detail later in chapter 8). New identifier 0.392 that used to index new character "m" can be computed by executing Algorithm 8 (will be discussed in next chapter).

7.2.1 Framework Description

It is supposed that all users in the network are capable to perform the same activities and the network could be structured or unstructured. Each user is capable to send and receive points in the network and any local modification by any user eventually received by all users in the network. Each user performs operations on a separate replica (*i.e.*, local copy) of the document. Operations may arrive and be executed at different sites in different orders, resulting in different final results. For the sake of efficiency and fault-tolerance, the content on the network is supposed to be replicated. This replication can be either total or partial in the sense that each user/peer can owns a replica partially or completely [WUM10]. A user can modify a replica at any time [Sun02] and it helps to attain high responsiveness and concurrency in the system. When a peer receives a modification, it replays it on its local replica. Thus, replicas are allowed to diverge in a short time. This kind of replication is known as optimistic replication [SS05] (or lazy replication). A collaborative editing system is considered to be correct [SJZ+98] if it satisfies the following conditions:

1. Causality: On every replica, the operations ordered by a precedence relation, in the sense of the Lamport's happened-before relation [Lam78], are executed in the same order. This property ensures the consistency of the execution orders of dependent operations during a cooperative editing session.

2. Convergence: When the same set of operations have been executed at all sites, all copies of the shared document are identical. The system converges to a stable state, where all the replicas are consistent (converge), *i.e.,* all replicas are up to date [PC04]. This property ensures the consistency of the final results at the end of a cooperative editing session.

3. Intention preservation: For any operation ϕ, the effects of executing ϕ at all sites are the same as the intention of ϕ, and the effect of executing ϕ does not change the effects of independent operations. the convergence property ensures the consistency of the final results at the end of a cooperative editing session. The following definition of operation intentions for textual documents is considered to be well-accepted:

- delete: A deleted line (character) must not appear in the document unless the inverse operation is not executed.

- insert: A line (character) inserted by user must appear to every user. The order relation between the document lines (characters) and a newly inserted line (character) must be preserved on every peer (as long as these lines / characters exist) [LL07].

We are interested to build decentralized collaborative editing system respecting CCI consistency model and supporting P2P constraints such as:

- Scalability: The system must be capable to handle the addition of users or objects without suffering a noticeable loss of performance [Neu94]. It means the system may consist of a finite large number of users / peers.

- Churn: Any user can be an active or lazy user, *i.e.*, a user can participate at any time.

- Unknown number of peers: The system must not subject to handle a fix number of users / peers, it should be compatible for unknown number of users / peers.

- Failures: It is possible that any peer can crash at any time without warning other peers. So the system must deal with this situation.

7.2.2 Editing a Document

Our approach to edit a document can be viewed in the CRDT (Commutative Replicated Data Type) framework [PMSL09] whose main idea is to provide a genuine commutativity between concurrent operations. CRDTs guarantee eventual consistency, provided that every site executes every operation in an order consistent with happens-before, the final state of replicas is identical at all sites. When the data type is an ordered list, such as a text document, commutativity between insertions in the list can be obtained with unique and totally ordered identifiers for each element. In our work, we consider a text document as a sequence of elements where an element can be a character, a line, a paragraph, etc.

Our approach is based on associating an identifier to each element. These identifiers must be unique, dense, and totally ordered. Our method for creating these identifiers is based on precision control, that falls under the floating point arithmetic environment and leads to an extension of the research in this direction. We prove the uniqueness of these identifiers and design algorithm to implement the idea of creation of these identifiers.

7.2.3 Modifying a Document

In the model that we proposed, users modify the shared document by inserting or deleting characters/lines. In performing this task, a creation or removal of corresponding identifiers occurs. Thousands of users are able to create millions of points with epsilon assigned to it, and by changing precisions. Each user can create points locally as well as based on the remote points during the exchange of points between the users in the network. All users in the network use the same criteria to compute points.

7.2.4 Mapping between elements and Identifiers

From a user's perspective, a document consists of a sequence of elements (characters, lines, etc.). If a new element is inserted, a portion of the sequence is to be dispersed to create the space for the new element. Similarly, if a element is deleted, a portion of the sequence is to be merged to reclaim the space. The editing system keeps the elements of the document in a selected data structure. The editing operation can be mapped to the creation/removal of corresponding identifiers.

- Insert a new element corresponds to the creation of new identifier.
- Delete an existing element corresponds to the removal of an identifier.

Therefore, if a user edits a document locally then an element corresponds to the creation or removal of an identifier.

To maintain the ordering information, for each element, its before and after element's identifiers need to be explicitly maintained. When reconstructing a document's content, the system has to follow these before and after identifiers iteratively and maps them into their corresponding elements. A set of totally ordered identifiers makes easy and efficient, the sequential reading and retrieving the elements. When reconstructing a document's content, the system has to follow these before and after identifiers iteratively and maps them into their corresponding elements. Consequently, the mapping between elements and identifiers impacts the performance of sequential reading of elements and their corresponding identifiers.

Chapter 8

Generating Identifiers

Contents

8.1 Creating Unique Identifiers 118
8.1.1 Precision Control Technique 118
8.1.2 Generating user/site identifiers (*USIDs*) 120
8.1.3 The Procedure 121
8.1.4 Explanation 124
8.1.5 Uniqueness of line/character identifiers (*LCIDs*) 125
8.1.6 Verifying uniqueness 127
8.2 Computing cardinality 127
8.2.1 Local cardinality 128
8.2.2 Global cardinality 128

In collaborative editing system (CES), due to concurrent editing conflicts, the communication must be carefully constructed to maintain data consistency. This leads towards the importance of generating unique identifiers that could be associated with characters or lines to be edited. From user's perspective a document is a sequence of characters. Therefore, a set of ephemeral keys which are prone to change with the modification of the document, is required to index the document. We address this issue by assigning immutable and ordered identifiers to the data items of a document. We introduce a precision control technique capable to generate an ordered set of unique real identifiers of the same pattern. To the best of our knowledge, before this, it was not recommended to use real number (see for instance [PMSL09]) as identifiers due to infinite precision. In this chapter, we explain, in detail, how to generate unique identifiers and how to compute cardinality of a set of identifiers. First section consists in explaining the precision control technique, rounding a value, generating user (site, peer) identifiers and line or character identifiers, related algorithms and explanations. Section 2 explains, the concept of local as well as global cardinality and computing the cardinalities.

8.1 Creating Unique Identifiers

In this section, we explain, Precision Control Technique (PCT), that we use to generate unique element (line, character) identifiers (*LCIDs*). Then we illustrate algorithms to generate identifiers with examples and figures.

Definition 14. *We define precision as the number of digits following the point of a value (rounded to decimal places/to significant digits), e.g., the precision of the values* 12.34600 *and* 12.345 *is* 5 *and* 3 *respectively.*

8.1.1 Precision Control Technique

To reduce existing errors and to provide strong foundation for collaborative systems, we introduce precision control technique (PCT) which is based on the following assumptions.

- $\mathbf{A}_1$: $''p_d''$ denotes the default precision that commonly used by programming language over which we perform computations.
  For example, in Maple, the precision used is controlled by the global variable "Digits" which has 10 as its default value and floating point arithmetic is done in decimal with rounding, so one can adjust default precision.

- $\mathbf{A}_2$: $''p_\epsilon''$ denotes the precision taken for small positive real numbers $'\epsilon'$ that acts as user/site identifiers (*USIDs*).

- $\mathbf{A}_3$: $''p_r''$ denotes the rounding precision and is kept less than both of p_ϵ and p_d.

Keeping in view, the above assumptions, we establish

$$p_r < p_\epsilon < p_d \tag{8.1}$$

inequality 8.1 and keep it as basic principle to generate unique identifiers ($LCIDs$) and to perform computations accordingly. Following (PCT), we assign a positive small real number $''\epsilon''$ to each user/site or peer (of the network $\aleph$) that acts as site identifier or user identifier (USID). To create identifiers, computation (calculation) are performed over the precision $''p_d''$ presented by gray bar in the Figure 8.1. and then values are rounded over precision $''p_r''$ presented by green bar in the Figure 8.1. The yellow bar presents precision $''p_\epsilon''$ taken to generate user/site identifiers (*USIDs*).

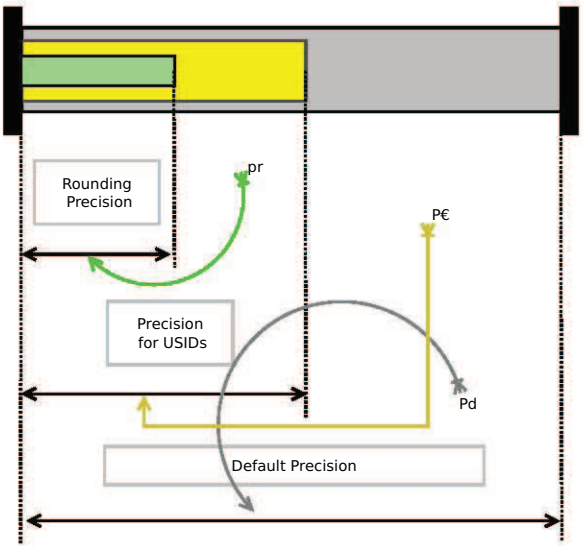


Figure 8.1: Precision control principle

Rounding values

To perform computation according to the basic principle (Inequality 8.1), we introduce Function 5 that rounds a given value according to definition 14.

Function 5:(Round a value)

Function: *Round a value*

Input: Value to be rounded, Desired precision
Output: Rounded Value

1 **begin**
2 Let dp := Desired precision;
3 x := Value to be rounded;
4 prec $\longleftarrow 10^{dp}$;
5 $y = \dfrac{round(x * \text{prec})}{\text{prec}}$;
6 Rounded value $\longleftarrow y$;
7 **return** *Rounded value*;
8 **end**

The function $''round(x)''$ (see line 5, Function 5) rounds an expression x to the nearest integer and the function $''Round\,a\,value(x,\, p_r)''$ rounds the decimal part of an expression to the $p_r th$ decimal place, *e.g.*, $round(-2.4)$ returns -2 and $''Round\,a\,value(15.0766647, 4)''$ returns 15.0767 performing computation using *Maple* 12. We denote $''Round\,a\,value(x,\, p_r)''$ by $\left|x\right|_{p_r}$, *i.e.*, value x rounded over precision p_r by the function $''Round\,a\,value''$.

8.1.2 Generating user/site identifiers (*USIDs*)

A number of choices available in selecting values that could be assigned as user/site identifiers but we observed that (in the existing techniques) there is no computational link between these identifiers and that for indexing characters/lines. We propose an approach to generate *USIDs* that are involved in generating line/character identifiers such that from an *LCID*, a *USID* could

Algorithm 6: How to generate *USIDs*

Algorithm: *Generate USIDs*
Result: user/site identifiers (*USIDs*)

```
1  begin
2      Let dp := Desired precision
3          𝕌 := Set of USIDs
4          w1 ⟵ dp + 2
5          w2 ⟵ 10^dp − 1
6      for i from 1 to w2 do
7          w3 ⟵ ⌊log10 (i)⌋
8          w4 ⟵ − (w1 + w3)
9          USID ⟵ i ∗ 10^w4
10         if member(USID, 𝕌) = false then
11             𝕌 := [ op(𝕌), USID ]
12             Arrange set 𝕌 in an order
13         end
14     end
15     return An Ordered Set of USIDs
16 end
```

be computed. In fact, we assign each user/site an epsilon (ϵ), a small positive real value that acts as user/site identifier and generate different *USIDs* of the form

$$\epsilon = 0.00\ldots0_{p_r+1}d_1d_2\ldots d_{p_r}$$

$$\text{where } p_r \geq 1,\ p_\epsilon \geq p_r + \mu,\ \mu \geq 2,\ \mu \in \mathbb{N}$$

Keeping this particular pattern makes it easy to deal with basic principle 8.1 and computing cardinality for *LCIDs*. Following this approach (as prescribed above), we can generate 9×10^w different epsilons with $w = p_r - 1$. Note that for $p_r \geq 1$, the minimum value of epsilon is $1 \times 10^{-(p_r+2)}$ and the maximum value is

$$\epsilon = 0.00\ldots0_{p_r+1}99\ldots9_{p_r}$$

A set of user/site identifiers generated by keeping $p_r = 1$ is shown in the table 8.1. To generate a set of different *USIDs*, we design Algorithm 6.

Set of user/site identifiers				
0.001	0.002	0.003	0.004	0.005
0.006	0.007	0.008	0.009	

Table 8.1: A set of user/site identifiers

The function $''member''$ (see line 10, Algorithm 6) verifies that either new computed $USID$ already exists in the set $\mathbb{U}$ or not. If it does not exist already then "*op*" (see line 11, Algorithm 6) collects it as a member of the set $\mathbb{U}$. Algorithm 6 returns a set of different epsilons under precision p_r. For instance, $Generate\ USIDs(4)$ will return a set of 9000 different epsilons computed under the precision $p_r = 4$ with minimum value of epsilon equals 1×10^{-6} and with maximum value equals 0.000009999. Command *op* is used to write input within an array. The *USIDs* maintain the property $|USID|_{p_r} = 0$.

8.1.3 The Procedure

It is known that the classical midpoint formula computes midpoint of two given points a and b as $(a+b)/2$ and if we apply it to compute all possible midpoints of an interval, then it works iteratively and continues to partition the interval infinitely. To compute each time different and finite possible midpoints for an interval, it requires some modifications. Let n ($n \in \mathbb{N}$) users compute all possible midpoints over an interval I, we are interested in, that:

- ⋆ Points computed for one user must be different from points computed for all others
- ⋆ Set of points computed for each user must be an ordered set
- ⋆ It should be possible to compute cardinality for the set of points.

To meet such requirements, we make some assumptions and modify the classical midpoint formula, for an interval $I = [\,a,\, b\,]$ with $0 \leq a < b$ such that $\forall\ x,\, y \in I$ with $x < y$, as given below

$$f(x,\, y) = x + \frac{y - x}{2} - \epsilon \tag{8.2}$$

where ϵ is one of the *USIDs*. Notice that the point computed in this way will not be equidistant from the points x and y (as explained in Figure 8.2) due to the conditions posed for computation. In all of our experimentations, we take the interval as $I = [\,0,\, 1\,]$ but we are not restricted to take only this interval. For two points $a_1,\ b_1$, we implement the formula (8.2) by designing Function 7, with $USID_{max} = max(USIDs)$. Function 7 takes two points as input and returns a new point by applying Formula (8.2) following the Precision Control Technique (PCT). We introduce additional conditions to assure the

order preservation and to get the computed points in the form of an ordered set. For this purpose we design Algorithm 8 that contains Function 7 as a part of it. In the Algorithm 8, the function "*nops*" (see line 5, 6 and 10) computes

Function 7: *middle*(How to compute new identifier)

Function: *middle*

Input: I=$[a_1, b_1]$, $USID$, $USID_{max}$, Desired precision

Output: *LCID*

```
1  begin
2      Let dp := Desired precision;
3          a ⟵ |a_1|_dp;
4          b ⟵ |b_1|_dp;
5      ∇ := |(b − a)/2|_dp;
6      if ∇ > USID_max then
7          point := |a + ∇|_dp − (USID);
8      end
9      return point;
10 end
```

number of elements in a set, for example, $nops([5, 6, 7])$ returns 3.

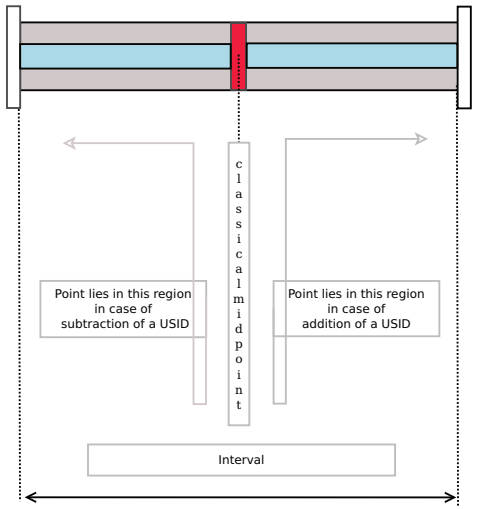


Figure 8.2: Position of an identifier in the interval

Algorithm 8: How to generate a set of *LCIDs*

Algorithm: *Generate LCIDs*

Input: $Interval$, $USID$, $USID_{max}$, Desired precision
Output: Set of unique identifiers (LCIDs)

```
1  begin
2      Let res := Interval
3          dp := Desired precision
4      tempres := Set of LCIDs
5      while nops(tempres) ≠ nops(res) do
6          if nops(tempres) ≠ 0 then
7              res ⟵ tempres
8              tempres ⟵ φ
9          end
10         for i ⟵ 1 to nops(res) − 1 do
11             LCID ⟵ middle(res[i], res[i + 1], USID, USID_max, dp)
12             tempres := [ op(tempres), res[i] ]
13             if (LCID ≠ res[i] and LCID ≠ res[i + 1]) then
14                 verify that
15                 if (LCID < res[i + 1] and LCID > res[i]) then
16                     tempres ⟵ [ op(tempres), LCID ]
17                 end
18             end
19         end
20         tempres ⟵ [ op(tempres), res[nops(res)] ]
21     end
22     return Set of LCIDs
23 end
```

Next, we summarize all steps that are performed above, in computing *LCIDs*, as follows.

Outlines

Let the interval $I = [\,a,\, b\,]$, $0 \leq a < b$, presents an empty document or a and b are two *LCIDs* for two corresponding characters/lines. To compute a new $LCID$ (say p) between a and b, we proceed as follows:

C1. Round a and b over the precision p_r

C2. Evaluate $\nabla = \dfrac{|b|_{p_r} - |a|_{p_r}}{2}$ and round it over p_r

C3. If $\nabla > USID_{max}$ then go to the next step

C4. $p := \Big||a|_{p_r} + \nabla\Big|_{p_r} - USID$

C5. Verify that $p \not\leq a$ and $p \not\geq b$.

8.1.4 Explanation

In whole of our experimentation, it is supposed that an empty shared document is mapped to an interval $I = [0, 1]$, as we stated before. Here, we explain the procedure for computing unique identifiers *LCIDs*. Figure 8.3

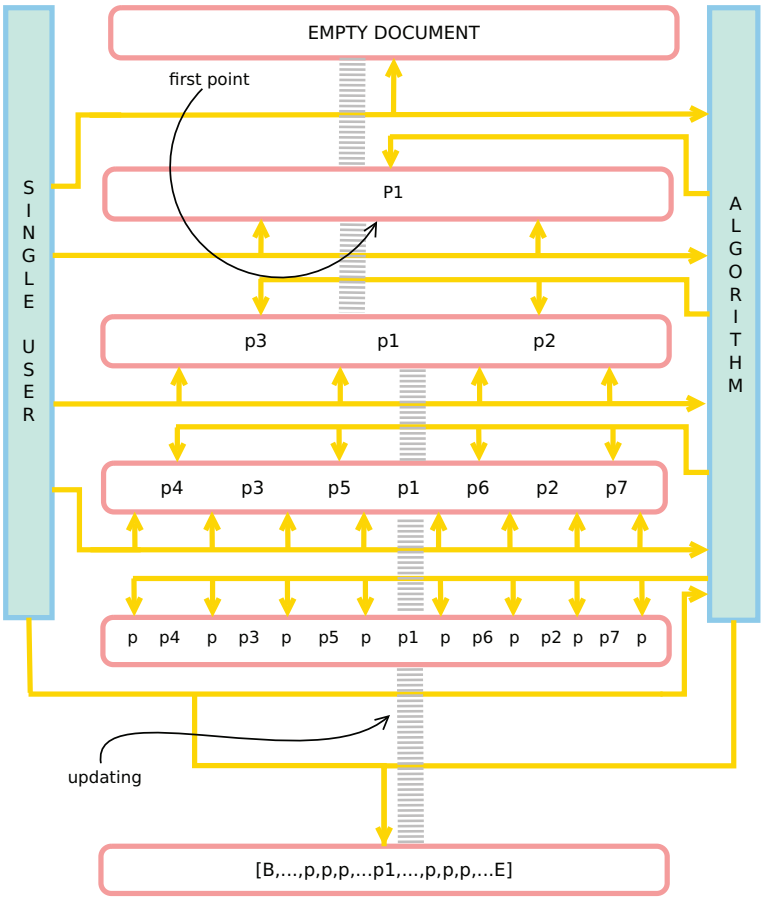


Figure 8.3: Insertion of points by single user

presents an overview of, how a single user inserts points, corresponding to unique identifiers, starting with an empty document. p_1 is the first identifier computed corresponding to first insertion by user. This first identifier partition the corresponding interval into two sub-intervals. Now, user has choices to insert new character on both sides of the first insertion. Therefore, let user inserted two more characters, one to the left side and other to the right side of the first insertion. This operation results in creation of two identifiers p_2 and p_3 to both sides of the first identifier p_1. Two more partitions occurred in the corresponding interval. After performing this operation, user inserted four more characters, each new character between each existing pair. This opera-

tion results in creation of four new corresponding identifiers p_4, p_5, p_6 and p_7 and partition of interval into further sub-intervals. Notice that each time we require the information only about the neighboring identifiers of new creation. This process continues and new points are presented by ps in the Figure 8.3 whereas an update operation is marked with 'updating'.

Next, we give an example to show that *LCIDs* computed by Algorithm 8 are different.

Example 23. *Let $p_r = 1$, and $p_\epsilon = 3$. Consider a set of nine different USIDs as given below*

$$\epsilon_1 = 0.001,\ \epsilon_2 = 0.002,\ \epsilon_3 = 0.003,\ \epsilon_4 = 0.004,\ \epsilon_5 = 0.005$$

$$\epsilon_6 = 0.006,\ \epsilon_7 = 0.007,\ \epsilon_8 = 0.008,\ \epsilon_9 = 0.009,\ \epsilon = 0.009$$

We compute all possible LCIDs for each user with different USIDs (by the Algorithm 8) as listed in the Table 8.2, in such a way that first row presents the points computed by first user, second row presents points computed by second user and so on. Notice that there is no common point between any two rows

User	***Unique Identifiers***									
U_1	0.099	0.199	0.299	0.399	0.499	0.599	0.699	0.799	0.899	0.999
U_2	0.098	0.198	0.298	0.398	0.498	0.598	0.698	0.798	0.898	0.998
U_3	0.097	0.197	0.297	0.397	0.497	0.597	0.697	0.797	0.897	0.997
U_4	0.096	0.196	0.296	0.396	0.496	0.596	0.696	0.796	0.896	0.996
U_5	0.095	0.195	0.295	0.395	0.495	0.595	0.695	0.795	0.895	0.995
U_6	0.094	0.194	0.294	0.394	0.494	0.594	0.694	0.794	0.894	0.994
U_7	0.093	0.193	0.293	0.393	0.493	0.593	0.693	0.793	0.893	0.993
U_8	0.092	0.192	0.292	0.392	0.492	0.592	0.692	0.792	0.892	0.992
U_9	0.091	0.191	0.291	0.391	0.491	0.591	0.691	0.791	0.891	0.991

Table 8.2: Set of unique identifiers

of the Table 8.2.

8.1.5 Uniqueness of line/character identifiers (*LCIDs*)

In this section, we prove that the algorithm 8 generates different *LCIDs* for different participants assigned with different identifiers.

Theorem 9. *Let two participants are assigned with two different USIDs (ϵ_i, ϵ_j such that $\epsilon_i < \epsilon_j$, for $0 < i < j$). Then for two operations performed by two participants respectively, two different LCIDs will be generated (over the interval $I = [a, b]$ with $0 \leq a < b$).*

Proof. Suppose that U_1 and U_2 be the two users, assigned with two different *USIDs*, ϵ_i and ϵ_j, compute two *LCIDs*, p_1 and p_2 by executing algorithm 8. For $x_i,\ x_j,\ y_i,\ y_j \in I = [\,a,\,b\,]$, and $v = p_r \in \mathbb{N}$, p_1 and p_2 can be written as

$$p_1 = \left| x_i + |\frac{y_i - x_i}{2}|_v \right|_v - \epsilon_i$$

$$p_2 = \left| x_j + |\frac{y_j - x_j}{2}|_v \right|_v - \epsilon_j$$

where $x_i = |x_i|_v,\ x_j = |x_j|_v,\ x_j = |x_j|_v,\ y_j = |y_j|_v$. For the sake of simplicity, suppose that

$$m_i = \left| x_i + |\frac{y_i - x_i}{2}|_v \right|_v$$

$$m_j = \left| x_j + |\frac{y_j - x_j}{2}|_v \right|_v$$

So, it is enough to prove that

$$|m_i - m_j| + (\epsilon_j - \epsilon_i) \neq 0 \tag{8.3}$$

Now, for $u_1,\ u_2 \in \mathbb{N}$

$$\epsilon_i = l_i.d_1^i d_2^i d_3^i \dots d_{u1}^i$$

$$\epsilon_j = l_j.d_1^j d_2^j d_3^j \dots d_{u1}^j$$

$$\epsilon_j - \epsilon_i = l_k.d_1^k d_2^k d_3^k \dots d_v^k d_{v+1}^k \dots d_{u2}^k$$

$$\epsilon_i \neq \epsilon_j$$

$$\therefore \text{ At least one of } d_{k1}^k \neq 0,\ v < k_1 \leq u_2 \leq u_1 \tag{8.4}$$

Let

$$\alpha = |m_i - m_j| = l_\alpha.d_1^\alpha d_2^\alpha d_3^\alpha \dots d_v^\alpha$$

$$\text{and} \quad \beta = \epsilon_j - \epsilon_i = l_\beta.d_1^\beta d_2^\beta d_3^\beta \dots d_v^\beta d_{v+1}^\beta \dots d_{u2}^\beta$$

$$\because\ v = p_r < p_\epsilon < p_d \text{ (basic principle (8.1))}$$

$$\implies p_\alpha < p_\beta \implies \alpha \neq \beta$$

$$\text{Also } \beta \neq 0 \text{ by (8.4)}$$

$$\text{Hence} \quad \alpha + \beta \neq 0 \text{ and (8.3) holds.}$$

□

8.1.6 Verifying uniqueness

To generate unique line/character identifier one must have to keep our proposed specific pattern of user/site identifiers and precision control technique. In the theorem 9, we prove the uniqueness even keeping only one zero after

Algorithm 9:

Algorithm: comparing different sets of *LCIDs*

Input: Desired precision, Interval
Output: *LCIDs* are unique

```
1  begin
2      Let dp := Desired precision
3            𝕌 := Set of USIDs
4            𝕀 := Interval
5            𝕌 ⟵ GenerateUSIDs(dp)
6            USID_max:= Max. value of USID in the set 𝕌
7                    M:= No. of elements in the set 𝕌
8      for kk ⟵ 1 to M do
9          USID ⟵ U[kk]
10         d[kk] ⟵ Generate LCIDs (𝕀, USID, USID_max, dp)
11     end
12     for i ⟵ 1 to M-1 do
13         MM:=No. of elements in d[i]
14         for j ⟵ 2 to MM-1 do
15             for k ⟵ i+1 to M do
16                 if member(d[i][j], d[k]) = true then
17                     USID_i:= USID corresponding to the set d[i]
18                     USID_k:= USID corresponding to the set d[k]
19                     if USID_i ≠ USID_j then
20                         d[i][j] is a common point between the sets d[i] and d[k]
21                         please verify the precision conditions
22                     end
23                 else
24                     No common points are there
25                 end
26
27             end
28         end
29     end
30     return No collision
31 end
```

decimal point but in generating line/character identifiers we keep more than one zero in that place. The reason is that, with only one zero, cardinality formulation is complicated that could be observed by drawing graph for the line/character identifiers generated by keeping only one zero. In addition to theorem 9, we provide a simple algorithm that make a comparison between line/character identifiers generated (see line 16) for different users and alerts if there is any mistake in selecting specific pattern to generate such identifiers.

8.2 Computing cardinality

Cardinality of a set is the "number of elements in the set". For example, the set $A = \{2, 4, 6\}$ contains 3 elements, and therefore cardinality of set A is 3.

Under Precision Control Technique (PCT), it is possible to compute cardinality of line/character identifiers *LCIDs*, generated by taking a particular precision.

Definition 15. *The possible number of identifiers (LCIDs) that could be created for a user (site) under particular precision is called local cardinality and is denoted by C_l.*

8.2.1 Local cardinality

Let γ denotes the length of the interval $I = [\,a,\, b\,]$, $0 \leq a < b$ then $\gamma = |b - a|$ with $\gamma > 0$. For the precision p_r, local cardinality for a single user can be computed as

$$C_l = \begin{cases} \gamma \times 10^{p_r} & \text{if } b > a \geq 0, \text{ for } a, b \in \mathbb{N} \\ (\gamma_1 \times 10^{p_r}) - 1 & \text{if } b > a > 0, \text{ for } a, b \in \mathbb{R} \\ & \text{and } \gamma_1 = |\gamma|_{p_r} \end{cases}$$

such that

$$p_r < p_\epsilon < p_d$$

USIDs are of the pattern as explained in section 8.1.2.

Example 24. *Suppose that $I_1 = [0,\, 1]$, $p_r = 3$, $USID_{min} = 0.00001$, and $USID_{max} = 0.0000999$. The local cardinality of a set of LCIDs, that a single user with $USID = 0.0000439$ can generate, is computed by $C_l = 1{*}10^3 = 1000$. Now, let the same user wants to compute local cardinality of LCIDs between two LCIDs (already computed by him). Let $I_2 = [0.7609561,\, 0.8139561]$ then $\gamma_1 = 0.053$ and $C_l = (0.053 * 10^3) - 1 = 52$.*

8.2.2 Global cardinality

We define global cardinality as follows.

Definition 16. *In a network of users/peers, the union of local cardinality computed by/at each user/peer is called global cardinality and is denoted by C_g.*

Computing global cardinality of a network depends on the local cardinalities, insertion/deletion, sending/receiving and the ways of exchange of *LCIDs* by the participants. We compute, here, global cardinality for a simple situation. Consider a network of n ($n \in \mathbb{N}$) users/peers and assume that each user/peer generates *LCIDs* locally and send to others. Each user/peer can also removes points that has already been created. Since two different users/peers generate different *LCIDs* (by Theorem 9), therefore, no duplication of *LCIDs* is possible during the exchange of elements between users/peers.

Bounds over cardinality

For a particular precision p_r, we take zero as the lower bound over local cardinality for a single user/peer and local cardinality computed by a single user for an empty document is considered to be an upper bound over local cardinality. We take local cardinality computed by a single user for an interval $I = [\, a, b \,], 0 \leq a < b$ (as explained in section 8.2.1) as a lower bound for global cardinality of a network of $n \in \mathbb{N}$ users/peers. Similarly, we take an upper bound over global cardinality as

$$C_g = \sum_{i=1}^{n} C_{l_i}$$

that corresponds to an ideal situation in which each user/peer achieves an upper bound over local cardinality.

Chapter 9

Properties and Analysis

Contents

9.1 Properties 132

9.1.1 Bijection property 132

9.1.2 Closure properties 133

9.2 Analysis 137

9.2.1 Comparison under Rounding Conditions 137

9.2.2 Comparison without Rounding Conditions . . . 138

9.2.3 Comparison under floating-point arithmetics . . 138

9.2.4 Effect of different rounding conditions 139

9.2.5 Assuring order preservation 147

Set of identifiers *LCIDs* generated by algorithm 8 (see chapter 8, section 8.1.3) holds some important properties that show advantages of the precision control indexing method. These properties provide assurance in generating identifiers based on remote points. The existence of closure properties confirm that there is no loss of identifier during exchange of points and all the identifiers generated don't escape from the interval allocated for all users in the network. Similarly, an analysis of the function 8.2 helps in exploring the usefulness of the function with rounding conditions and explains its limitations.
In this chapter, we prove some important properties that hold on the set of identifiers *LCIDs* generated by the algorithm 8. In section 9.2, we present an analysis of function 8.2 by comparing it with classical midpoint formula under different situations. Section 9.2.4 presents effect of different rounding conditions over the function (function 8.2) used in algorithm 8. Section 9.2.5 describes how our approach assure order preservation in exchange of points across the network particularly when identifies are required to generate on the basis of identifiers of remote points.

9.1 Properties

In this section, we prove that set of identifiers *LCIDs* holds bijection property as well as some closure properties.

9.1.1 Bijection property

Proposition 5. *Let S' be the set defined over the interval $I = [\,a,\, b\,], 0 \leq a < b$, as*

$$S' = \{\frac{n'}{10^{p_r}} \mid n' \in \mathbb{N},\, a < n' \leq \beta,\, \beta = \gamma * 10^{p_r}\}$$

where γ is the length of the interval I with $|\gamma|_{p_r} > 0$. For $1 \leq k \leq n,\ n \in \mathbb{N}$, let ϵ_k be one of the USIDs then the function $f_k : S' \longrightarrow S''$ defined by $f_k(x') = x' - \epsilon_k, \forall\, x' \in S'$, is a bijection.

Proof. The domain of the function f_k is the set

$$S' = \{\frac{n'}{10^{p_r}} \mid n' \in \mathbb{N},\, a < n' \leq \beta,\, \beta = \gamma * 10^{p_r}\}$$

and the range of the function f_k is the set

$$S'' = \{x' - \epsilon_k \mid \forall\, x' \in S',\, p_\epsilon \geq p_r + \mu,\ \mu \geq 2\}$$

Now, we have to prove that the given function f_k is both injective and surjective

- Injection
  Suppose that
  $$x'_1 \neq x'_2,\ \forall\, x'_1,\ x'_2 \in S'$$
  then
  $$f_k(x'_1) = x'_1 - \epsilon_k \text{ and } f_k(x'_2) = x'_2 - \epsilon_k$$
  Let
  $$f_k(x'_1) = f_k(x'_2)$$
  $$\implies x'_1 = x'_2$$
  A contradiction to the supposition that $x'_1 \neq x'_2$. Thus
  $$f_k(x'_1) \neq f_k(x'_2)$$

- Surjection

For surjection, we have to prove that

$$\exists\, x' \in S', \text{ such that } x'' = f_k(x'),\ \forall x'' \in S''$$

$$\because\ \ x' = x'' + \epsilon_k$$

$$\therefore\ \ f_k(x'' + \epsilon_k) = x'' + \epsilon_k - \epsilon_k = x''$$

$$\implies f_k \text{ is a surjective functions}$$

Hence for $1 \leq k \leq n$, f_k presents a family of bijective functions. □

The bijection property as described in proposition 5 helps in verifying the cardinality formula proposed in previous chapters. Set S'' contains *LCIDs* similar generated by the algorithm 8. For example, for the interval [0, 1], we can generate values

$$[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]$$

It generates *LCIDs* by subtracting corresponding *USIDs*, under chosen rounding precision.

Definition 17. *We denote, the set of USIDs computed by user U_k with $USID(\epsilon_k)$ over the interval $[a, b]$, $0 \leq a < b$, by S_k, where $1 \leq k \leq n$, $n \in \mathbb{N}$.*

Corollary 2. *For the interval $I = [a, b]$, $0 \leq a < b$, two sets S'' and S_k are equal.*

Proof. For a $USID = \epsilon_k$, the set S_k can be written as

$$S_k = \{x_k \mid x_k = |x_k|_{p_r} - \epsilon_k\}$$

Let $\acute{x_k} \in S_k$ then $\acute{x_k} = |\acute{x_k}|_{p_r} - \epsilon_k$. Now $|\acute{x_k}|_{p_r} \in \acute{S}$, where

$$S' = \{\frac{n'}{10^{p_r}} \mid n' \in \mathbb{N}, 1 \leq n' \leq \beta, \beta = \gamma \times 10^{p_r}\}$$

But $|\acute{x_k}|_{p_r} - \epsilon_k \in S''$ by Proposition 5.
Therefore S_k and S'' are equal sets generated over the same interval. □

9.1.2 Closure properties

In this section, we prove some important closure properties.

First Property

Lemma 5. *Let for $1 \leq k \leq n$, $n \in \mathbb{N}$, $\epsilon_k \in \{\epsilon_1, \epsilon_2, \epsilon_3, \ldots, \epsilon_{n-1}, \epsilon_n\}$ be the $USID$ for user U_k and S_k be the set of LCIDs generated for the user U_k over the interval $I = [a, b]$, $0 \leq a < b$. Then for $x, y \in S_k$, $x < y$,*

$$\left| |x|_{p_r} + \left| \frac{|y|_{p_r} - |x|_{p_r}}{2} \right|_{p_r} \right|_{p_r} - \epsilon_k \in S_k$$

Proof. Let $x, y \in S_k$ then

$$x = |x|_{p_r} - \epsilon_k, \text{ and } |x|_{p_r} \in \acute{S}$$

$$y = |y|_{p_r} - \epsilon_k, \text{ and } |y|_{p_r} \in \acute{S}$$

where

$$S' = \{\frac{n'}{10^{p_r}} \mid n' \in \mathbb{N}, 1 \leq n' \leq \beta, \beta = \gamma \times 10^{p_r}\}$$

Suppose that

$$\nabla = \left| \frac{|y|_{p_r} - |x|_{p_r}}{2} \right|_{p_r} > (USID)\max$$

That is, condition C5. (section 23) is satisfied. It gives $\nabla \in \acute{S}$,
Now

$$|x|_{p_r} < \left| |x|_{p_r} + \nabla \right| \leq |y|_{p_r}$$

But

$$\left| |x|_{p_r} + \nabla \right|_{p_r} \in \acute{S}$$

Therefore, by definition 17.

$$\left| |x|_{p_r} + \nabla \right|_{p_r} - \epsilon_k \in S_k$$

□

This property proves that over the set of identifiers generated for a user, any new identifier generated (for the same user) on the basis of two identifiers that have already generated, this new identifier lies in the same set.

Second Property

Lemma 6. *Let S_i, S_j, and S_k be the set of LCIDs generated for users U_i, U_j and U_k with USIDs, ϵ_i, ϵ_j and ϵ_k over the interval $I = [a, b]$, $0 \leq a < b$ respectively. Then for any two points $x \in S_i$, $y \in S_j$, $x < y$ and for $\epsilon_j < \epsilon_k$.*

$$\left| |x|_{p_r} + \left| \frac{|y|_{p_r} - |x|_{p_r}}{2} \right|_{p_r} \right|_{p_r} - \epsilon_k \in S_k$$

Proof. Let $x \in S_i$ and $y \in S_j$ then

$$x = |x|_{p_r} - \epsilon_i, \text{ and } |x|_{p_r} \in \acute{S}$$

$$y = |y|_{p_r} - \epsilon_j, \text{ and } |y|_{p_r} \in \acute{S}$$

Also, the sets S_i, S_j, and S_k can be written as

$$S_i = \{x' - \epsilon_i \mid \forall\, x' \in S',\, p_\epsilon \geq p_r + \mu,\, \mu \geq 2\}$$

$$S_j = \{x' - \epsilon_j \mid \forall\, x' \in S',\, p_\epsilon \geq p_r + \mu,\, \mu \geq 2\}$$

$$S_k = \{x' - \epsilon_k \mid \forall\, x' \in S',\, p_\epsilon \geq p_r + \mu,\, \mu \geq 2\}$$

where

$$S' = \{\frac{n'}{10^{p_r}} \mid n' \in \mathbb{N},\, 1 \leq n' \leq \beta,\, \beta = \gamma \times 10^{p_r}\}$$

Suppose that

$$\nabla = \left| \frac{|y|_{p_r} - |x|_{p_r}}{2} \right|_{p_r} > \epsilon, \text{ with, } \epsilon = max(\epsilon_i, \epsilon_j, \epsilon_k)$$

$$\implies |x|_{p_r} \neq |y|_{p_r}$$

Otherwise, condition C5. (section 23) does not hold.
Now, let

$$\acute{p} = \Big||x|_{p_r} + \nabla\Big|_{p_r}$$

then

$$|x|_{p_r} < \acute{p} \leq |y|_{p_r}$$

Let

$$\acute{p} = |y|_{p_r}$$

then

$$\acute{p} \in \acute{S}$$

Now, since

$$\epsilon_j < \epsilon_k$$

Therefore,

$$\acute{p} - \epsilon_k < \acute{p} - \epsilon_j \text{ and } \acute{p} - \epsilon_k \in S_k$$

□

This property proves that if an identifier is generated for first user and another identifier is generated for second user, then an identifier generated for third user lies within the set of identifier generated for third user.

Third Property

Lemma 7. *Let S_i be the set of LCIDs generated by user (or at site) U_i with $USID = \epsilon_i$ over the interval $I = [\,a,\, b\,]$, $0 \leq a < b$. Then for any two points $x,\, y \in S_i$, $x < y$ and for $\epsilon_i < \epsilon_k$.*

$$\left||x|_{p_r} + \left|\frac{|y|_{p_r} - |x|_{p_r}}{2}\right|_{p_r}\right|_{p_r} - \epsilon_k \in S_k$$

Proof. Let

$$p_{i,1} = \left||x|_{p_r} + \left|\frac{|y|_{p_r} - |x|_{p_r}}{2}\right|_{p_r}\right|_{p_r}$$

be the first rounded value computed by user U_i over the interval I. Since, each user starts computing values over the same interval I keeping same conditions of the algorithm. Therefore, for, $1 \leq i \leq n$, we have

$$p_{1,1} = p_{2,1} = p_{3,1} = \ldots = p_{i,1} = p_{(i+1),1} = \ldots = p_{n,1}$$

But by Theorem 9.

$$p_{1,1} - \epsilon_1 \neq p_{2,1} - \epsilon_2 \neq \ldots \neq p_{i,1} - \epsilon_i$$

$$\neq p_{(i+1),1} - \epsilon_{i-1} \neq \ldots = p_{n,1} - \epsilon_n$$

By definition 17,

$$p_{1,1} - \epsilon_1 \in S_1,\ p_{2,1} - \epsilon_2 \in S_2, \ldots, p_{n,1} - \epsilon_n \in S_n$$

Hence

$$\left| |x|_{p_r} + \left| \frac{|y|_{p_r} - |x|_{p_r}}{2} \right|_{p_r} \right|_{p_r} - \epsilon_k \in S_k$$

□

This property proves that if a set of identifiers is generated for first user then a new identifier generated for second user (based on the set of identifiers generated for first user), belongs to the set of identifiers generated for second user.

Fourth property

Lemma 8. *For* $1 \leq k \leq n$, $n \in \mathbb{N}$, *let* U_k *be the set of users/sites and* $\mathbb{S} = \bigcup_{k=1}^{n} S_k$, *with* $x < y$ *be the set of USIDs generated by all* U_k*/sites then for* $x,\ y \in \mathbb{S}$ *and* $\epsilon_k \in \{\epsilon_1,\ \epsilon_2,\ \epsilon_3, \ldots, \epsilon_{n-1},\ \epsilon_n\}$, *the point*

$$\left| |x|_{p_r} + \left| \frac{|y|_{p_r} - |x|_{p_r}}{2} \right|_{p_r} \right|_{p_r} - \epsilon_k \in S_k$$

Proof. Suppose that

$$\nabla = \left| \frac{|y|_{p_r} - |x|_{p_r}}{2} \right|_{p_r} > \epsilon,$$

with

$$USID_{max} = max(\epsilon_1,\ \epsilon_2,\ \epsilon_3, \ldots, \epsilon_{n-1},\ \epsilon_n)$$

Now, let $x,\ y \in S_k$ then, by Lemma (5).

$$\left| |x|_{p_r} + \left| \frac{|y|_{p_r} - |x|_{p_r}}{2} \right|_{p_r} \right|_{p_r} - \epsilon_k \in S_k$$

Let $x \in S_i$ and $y \in S_j$ then, by Lemma (6).

$$\left| |x|_{p_r} + \left| \frac{|y|_{p_r} - |x|_{p_r}}{2} \right|_{p_r} \right|_{p_r} - \epsilon_k \in S_k$$

Finally, let $x,\ y \in S_i$ then, by Lemma (7).

$$\left| |x|_{p_r} + \left| \frac{|y|_{p_r} - |x|_{p_r}}{2} \right|_{p_r} \right|_{p_r} - \epsilon_k \in S_k$$

Since $S_k \subseteq \mathbb{S}$, therefore

$$\left| |x|_{p_r} + \left| \frac{|y|_{p_r} - |x|_{p_r}}{2} \right|_{p_r} \right|_{p_r} - \epsilon_k \in \mathbb{S}$$

Hence the proof. □

9.2 Analysis

In this section, we make an analysis for the function 8.2 to explore some of its advantages. The classical midpoint formula excludes condition C4. (see for detail, chapter 8, section 8.1.3).

C4. $p := \Big||a|_{p_r} + \nabla\Big|_{p_r} - USID$

and rounding restrictions described in Outlines (chapter 8, section 8.1.3). We make a comparison between two formulas as follows:

9.2.1 Comparison under Rounding Conditions

Under rounding conditions formula 8.2 compute more points as compared to the classical midpoint formula.

Example 25. *Consider the interval* $I = [0,\ 0.001]$ *with (user-identifier)* $= 0.00001$*, epsilon* $= 0.00001$ *and* $p_r = 3$*. We are able to insert one point inside the interval* I *using formula 8.2 as given below,*

$$I = [0,\ 0.00099,\ 0.001]$$

with $\nabla = 0.0005$ *and* $\nabla > epsilon$*, but we cannot insert any point inside the interval* I *using classical midpoint formula due to the fact that rounding* ∇ *over* $p_r = 3$ *returns* $\nabla = 0.001$ *which is the end point of the interval* I*.*
Similarly for the interval $I = [0,\ 0.002]$*, using formula 8.2, we can compute one more point over the interval* I *as compared to classical midpoint formula, with* $p_r = 3$*, epsilon* $= 0.00001$ *and (user-identifier)* $= 0.00001$*.*

$$[0,\ 0.00099,\ 0.00199,\ 0.002]$$

$$[0,\ 0.001,\ 0.002]$$

With different values of epsilons and rounding scheme (as we proposed in Theorem 9), formula 8.2 generate different points (*see* Table: 8.2), while the classical midpoint formula does not do the same under the same environment (*see* Table: 9.1). Nine users assigned with different *USIDs* as given below:

$$\epsilon_1 = 0.0100,\ \epsilon_2 = 0.0200,\ \epsilon_3 = 0.0300,\ \epsilon_4 = 0.0400$$
$$\epsilon_5 = 0.0500,\ \epsilon_6 = 0.0600,\ \epsilon_7 = 0.0700,\ \epsilon_8 = 0.0800$$
$$\epsilon_9 = 0.0900,\ \epsilon = 0.0900$$

participate. The identifiers are listed in table 9.1.

User	***Similar Identifiers***								
U_1	0.1000	0.2000	0.3000	0.4000	0.5000	0.6000	0.7000	0.8000	0.9000
U_2	0.1000	0.2000	0.3000	0.4000	0.5000	0.6000	0.7000	0.8000	0.9000
U_3	0.1000	0.2000	0.3000	0.4000	0.5000	0.6000	0.7000	0.8000	0.9000
U_4	0.1000	0.2000	0.3000	0.4000	0.5000	0.6000	0.7000	0.8000	0.9000
U_5	0.1000	0.2000	0.3000	0.4000	0.5000	0.6000	0.7000	0.8000	0.9000
U_6	0.1000	0.2000	0.3000	0.4000	0.5000	0.6000	0.7000	0.8000	0.9000
U_7	0.1000	0.2000	0.3000	0.4000	0.5000	0.6000	0.7000	0.8000	0.9000
U_8	0.1000	0.2000	0.3000	0.4000	0.5000	0.6000	0.7000	0.8000	0.9000
U_9	0.1000	0.2000	0.3000	0.4000	0.5000	0.6000	0.7000	0.8000	0.9000

Table 9.1: Similar Identifiers

I	p_e	ϵ_i	ϵ	$Cd(f_c)$	$Cd(f_1)$	$Cd(f_1) - Cd(f_c)$
[0, 0.01]	3	0.003	0.003	3	33	30
[0, 0.01]	4	0.0003	0.0003	33	243	210
[0, 0.01]	5	0.00003	0.00003	257	2216	1959
[0, 0.01]	6	0.000003	0.000003	2049	19380	17331
[0, 0.0001]	7	0.0000003	0.0000003	257	2216	1959
[0, 0.0001]	8	0.00000003	0.00000003	2049	19580	17371
[0, 0.0001]	9	0.000000003	0.000000003	32769	165039	132270

Table 9.2: Cardinality Comparison

9.2.2 Comparison without Rounding Conditions

Let $Cd(f_1)$ and $Cd(f_c)$ denotes the cardinality of points inserted by formula 8.2 and classical formula respectively. With different values of epsilon and length of the interval I, we observe a notable difference in $Cd(f_1)$ and $Cd(f_c)$ as given in the Table 9.2. We observed that, on increasing the precision of epsilons, the cardinality increases more for the points computed by the formula 8.2 as compare to the cardinality computed by the classical formula.

9.2.3 Comparison under floating-point arithmetics

We also prefer to use the fraction

$$x_i + \frac{x_{i+1} - x_i}{2} \text{ over } \frac{x_{i+1} + x_i}{2}$$

due to the fact that midpoint computed by later one may not lies within the interval under certain restriction on floating-point arithmetics. For example, for an interval $I = [0.982, 0.987]$, if we are restricted to perform calculation on a machine with three decimal digit chopped floating point arithmetic, then

$$x + y = 1.96 \text{ and } \frac{x + y}{2} = 0.980 \notin I$$

9.2.4 Effect of different rounding conditions

In this section, we analyse function 8.2 with different rounding conditions so that the effect over the set of identifiers *LCIDs* can be observed. Consider a situation in which users/sites in the network participate in an order to compute *LCIDs*. Each user in the network computes *LCIDs* locally and sends it to all other users/sites. For a set of $USIDs$ $\{\epsilon_1, \epsilon_2, \epsilon_3, \ldots, \epsilon_{n-1}, \epsilon_n\}$, let $USID_{\min} = min(\epsilon_k)$, where $1 \leq k \leq n$, for $n \in \mathbb{N}$,
To analyse function 8.2, we modify rounding conditions (see line 2 to line 4 in the modified form of function 7 as given below).

Modification-I

```
   Input: I=[a_1, b_1], USID, USID_max, Precision(p_r)
1  begin
2  |  a ← a_1;
3  |  b ← b_1;
4  |  ∇ := |(b − a)/2|_{p_r} ;
5  |  if ∇ > USID_max then
6  |  |  point := |a + ∇|_{p_r} − (USID);
7  |  end
8  |  return point;
9  end
```

We also exclude condition C5. of the outlines (section 8.1.3, page 123) to compute LCIDs.
We consider the following two situations.

1. If the set of *USIDs* is in ascending order
2. If the set of *USIDs* is in descending order

Computing *LCIDs* keeping *USIDs* in ascending order

In this environment, we suppose that the user (site) assigned with $USID_{\min}$ starts computing and sends points to others such that the computation of *USIDs* occurs in a way that user U_{i+1} starts computation after user U_i. In this case the set of *USIDs* is in ascending order, *i.e.* $\{\epsilon_1, \epsilon_2, \epsilon_3, \ldots, \epsilon_{n-1}, \epsilon_n\}$. We explain the situation by the following example.

Example 26. *Suppose that* $I = [\,0.0, 0.02\,]$ *be an interval and user* U_1 *with* $USID = 0.0001$ *starts computing LCIDs keeping precision* $p_r = 2$. *Then the set of USIDs generated corresponding to the modifications performed by user* U_1 *is given below.*

$$S_1 = [\,0.0, 0.0099, 0.0199, 0.02\,]$$

Now, U_1 sends LCIDs to other users/sites so that updates are executed at each site of the network. Now, user U_2 with $USID = \epsilon_2 = 0.0002$ has to compute new USIDs based on the points sent by U_1. The set of USIDs computed by U_2 is

$$S_2 = [\,0.0,\ 0.0099,\ 0.0198,\ 0.0199,\ 0.02\,]$$

U_2 sends LCIDs to next users.
Similarly, user U_3 with $USID = \epsilon_3 = 0.0003$ has to compute USIDs based on the points that it received. But U_3 is not able to compute any point because the ∇ condition is not valid. Consider, for example,

$$\nabla = \left|\frac{0.0198 - 0.0099}{2}\right|_2 = \left|0.00495\right|_2 \not> USID_{max}$$

Hence, for an interval $I = [\,0,\ 1\,]$ and a set of nine users/sites assigned a set of *USIDs* as given below

$$[\,0.001,\ 0.002,\ 0.003,\ 0.004,\ 0.005,\ 0.006,\ 0.007,\ 0.008,\ 0.009\,]$$

User at site with $USID = 0.001$ computes 10 *USIDs*

$$[\,0,\ 0.099,\ 0.199,\ 0.299,\ 0.399,\ 0.499,\ 0.599,\ 0.699,\ 0.799,\ 0.899,\ 0.999,\ 1\,]$$

and sends to next users/sites. Now user U_2 with $USID = 0.002$ computes 9 more LCIDs
but any other user could not be able to increase the global cardinality that is

A final set after the participation of each user					
0.000	0.099	0.198	0.199	0.298	
0.299	0.398	0.399	0.498	0.499	
0.598	0.599	0.698	0.699	0.798	
0.799	0.898	0.899	0.998	0.999	1.000

Table 9.3: Set of Identifiers (LCIDs)

21 as shown in the Table 9.3 We observe that

- Only two users are able to compute the points but not all.
- Final set of *USIDs* is an ordered set and *USIDs* are in ascending order.

Computing *LCIDs* keeping *USIDs* in descending order

In this environment, the user (site) assigned with $USID_{max}$ starts computing and sending points to others, in a way that user U_i starts computation after user U_{i+1}. In this case the set of *USIDs* is in decreasing order, *i.e.*, $\{\epsilon_n,\ \epsilon_{n-1},\ \epsilon_{n-2}, \ldots, \epsilon_2,\ \epsilon_1\}$.

Example 27. *For the precision $p_r = 2$, consider a set of* 90 *users such that user U_{90} with $USID = \epsilon_{90} = 0.00099$ starts computing points for the interval $I = [\,0,\, 0.02\,]$,*

$$S_{90} = [\,0,\, 0.00901,\, 0.01901,\, 0.02\,]$$

and U_{90} sends LCIDs to the next user U_{89} with $USID = \epsilon_{89} = 0.00098$. User U_{89} computes one point based on the points sent by U_{90}

$$[\,0,\, 0.00901,\, 0.01902,\, 0.01901,\, 0.02\,]$$

and sends to next users. Now, user U_{88} with $USID = \epsilon_{88} = 0.00097$ computes one point because the ∇ condition is valid. Consider two consecutive points, for instance,

$$[\,0.01902,\, 0.00901\,]$$

$$\nabla = \left|\frac{0.00901 - 0.01902}{2}\right|_2 = 0.01 > USID_{max}$$

Therefore, next user can compute one more point as given below.

$$[\,0.,\, 0.00901,\, 0.01903,\, 0.01902,\, 0.01901,\, 0.02\,]$$

Similarly, we observe that, all other users can compute LCIDs and gives the global cardinality as 93 *including the end points of the interval.*

Thus for an interval $I = [\,a,\, b\,]$, and a set of nine users with a set of nine USIDs as given below

$$[\,0.001,\, 0.002,\, 0.003,\, 0.004,\, 0.005,\, 0.006,\, 0.007,\, 0.008,\, 0.009\,]$$

User with $USID = 0.009$ will compute 10 points, while all other users compute 9 points as given in the Table 9.4. Thus the global cardinality for the given interval is 84. We observe that

Similar Identifiers									
0.0	0.091	0.199	0.198	0.197	0.196	0.195	0.194	0.193	0.192
0.191	0.299	0.298	0.297	0.296	0.295	0.294	0.293	0.292	0.291
0.399	0.398	0.397	0.396	0.395	0.394	0.393	0.392	0.391	
0.499	0.498	0.497	0.496	0.495	0.494	0.493	0.492	0.491	
0.599	0.598	0.597	0.596	0.595	0.594	0.593	0.592	0.591	
0.699	0.698	0.697	0.696	0.695	0.694	0.693	0.692	0.691	
0.799	0.798	0.797	0.796	0.795	0.794	0.793	0.792	0.791	
0.899	0.898	0.897	0.896	0.895	0.894	0.893	0.892	0.891	
0.999	0.998	0.997	0.996	0.995	0.994	0.993	0.992	0.991	1.0

Table 9.4:

- All users are able to compute the points.
- The final set of Inserted points is not in order.

Modification-II

To analyse function 8.2, we modify rounding conditions (see line 2 to line 6 in the modified form of function 7 as given below). Here, we also exclude

```
   Input: I=[a_1, b_1], USID, USID_max, Precision(p_r)
1  begin
2  |  a ← a_1;
3  |  b ← b_1;
4  |  ∇ := (b − a)/2;
5  |  if ∇ > USID_max then
6  |  |  point := |a + ∇|_{p_r} − (USID);
7  |  end
8  |  return point;
9  end
```

condition C5. of the outlines (section 8.1.3, page 123) to compute LCIDs.

Computing LCIDs keeping USIDs in Ascending Order

Consider the following example, to observe the effect of above modification of the Function 7.

Example 28. *Let* $I = [\,0,\, 0.3\,]$ *be the interval and user* U_1 *with* $USID = \epsilon_1 = 0.001$ *starts computing USIDs under precision* $p_r = 1$*, Then*

$$S_1 = [\,0,\, -0.001,\, 0.099,\, 0.199,\, 0.3\,]$$

U_1 *sends LCIDs to the next user* U_2 *with* $USID = \epsilon_2 = 0.002$*. User* U_2 *computes points based on the points sent by* U_1

$$S_2 = [\,0,\, -0.001,\, -0.002,\, 0.099,\, 0.098,\, 0.199,\, 0.198,\, 0.3\,]$$

and sends to next users. Now, user $U3$ *with* $USID = \epsilon_3 = 0.003$ *computes points because the* ∇ *Condition is valid.*

$$[0., -0.001, -0.002, -0.003, 0.099, 0.098, 0.097, 0.199, 0.198, 0.197, 0.3]$$

At the completion of the process we have a set of 29 *LCIDs as listed in the Table 9.5*

Similarly, under precision $p_r = 1$, for an interval $I = [0,\ 1]$ and a set of nine users as given below

$$[0.001,\, 0.002,\, 0.003,\, 0.004,\, 0.005,\, 0.006,\, 0.007,\, 0.008,\, 0.009]$$

Each user is able to compute 10 points. So the global cardinality for the given interval will be 92 as shown in the Table 9.6. We observe that

- Each user is able to insert points due to the modified rounding scheme
- Each patch of *USIDs* is in decreasing order.

A final set after the participation of each user					
0	−0.001	−0.002	−0.003	−0.004	−0.005
−0.006	−0.007	−0.008	−0.009	0.099	0.098
0.097	0.096	0.095	0.094	0.093	0.092
0.091	0.199	0.198	0.197	0.196	0.195
0.194	0.193	0.192	0.191	0.3	

Table 9.5: Set of Identifiers

A final set after the participation of each user									
0	−0.001	−0.002	−0.003	−0.004	−0.005	−0.006	−0.007	−0.008	−0.009
0.099	0.098	0.097	0.096	0.095	0.094	0.093	0.092	0.091	
0.199	0.198	0.197	0.196	0.195	0.194	0.193	0.192	0.191	
0.299	0.298	0.297	0.296	0.295	0.294	0.293	0.292	0.291	
0.399	0.398	0.397	0.396	0.395	0.394	0.393	0.392	0.391	
0.499	0.498	0.497	0.496	0.495	0.494	0.493	0.492	0.491	
0.599	0.598	0.597	0.596	0.595	0.594	0.593	0.592	0.591	
0.699	0.698	0.697	0.696	0.695	0.694	0.693	0.692	0.691	
0.799	0.798	0.797	0.796	0.795	0.794	0.793	0.792	0.791	
0.899	0.898	0.897	0.896	0.895	0.894	0.893	0.892	0.891	1

Table 9.6: Set of Identifiers

Computing *LCIDs* keeping *USIDs* in descending order

With a set of *USIDs* in decreasing order, consider the same interval $I = [\,0,\, 0.3\,]$ and precision $p_r = 1$, as taken in Example 28.

Example 29. *Suppose* $p_r = 1$, *user* u_1 *at* $\epsilon_1 = 0.009$ *computes points for the interval* [0, 0.3],

$$[\,0,\, -0.009,\, 0.091,\, 0.191,\, 0.3\,]$$

and sends to the next user u_2 *at* $\epsilon_2 = 0.008$. *User* u_2 *computes points on the basis of points received from* u_1

$$[\,0,\, -0.009,\, -0.008,\, 0.091,\, 0.092,\, 0.191,\, 0.192,\, 0.3\,]$$

and sends to next users. Now, user u_3 *at* $\epsilon_3 = 0.007$ *computes points because the* ∇ *Condition is valid.*

$$[\,0.0,\, -0.009,\, -0.008,\, -0.007,\, 0.091,\, 0.092,\, 0.093,\, 0.191,\, 0.192,\, 0.193,\, 0.3\,]$$

we compute the LCIDs as given in the Table 9.7.

A final set after the participation of each user					
0	−0.009	−0.008	−0.007	−0.006	−0.005
−0.004	−0.003	−0.002	−0.001	0.091	0.092
0.093	0.094	0.095	0.096	0.097	0.098
0.099	0.191	0.192	0.193	0.194	0.195
0.196	0.197	0.198	0.199	0.3	

Table 9.7: Set of Identifiers

Similarly for the interval $I = [0, 1]$ with a set of nine users/sites assigned a set of *USIDs* as given below

$$[\,0.001,\ 0.002,\ 0.003,\ 0.004,\ 0.005,\ 0.006,\ 0.007,\ 0.008,\ 0.009\,]$$

User at $epsilon = 0.009$ will compute 10 points, and all other users also computes 10 points. So the global cardinality for the given interval will be 92 as listed in the Table 9.8.

A final set after the participation of each user									
0	−0.009	−0.008	−0.007	−0.006	−0.005	−0.004	−0.003	−0.002	−0.001
0.091	0.092	0.093	0.094	0.095	0.096	0.097	0.098	0.099	
0.191	0.192	0.193	0.194	0.195	0.196	0.197	0.198	0.199	
0.291	0.292	0.293	0.294	0.295	0.296	0.297	0.298	0.299	
0.391	0.392	0.393	0.394	0.395	0.396	0.397	0.398	0.399	
0.491	0.492	0.493	0.494	0.495	0.496	0.497	0.498	0.499	
0.591	0.592	0.593	0.594	0.595	0.596	0.597	0.598	0.599	
0.691	0.692	0.693	0.694	0.695	0.696	0.697	0.698	0.699	
0.791	0.792	0.793	0.794	0.795	0.796	0.797	0.798	0.799	
0.891	0.892	0.893	0.894	0.895	0.896	0.897	0.898	0.899	1

Table 9.8: Set of Identifiers

- Notice that, if we ignore the negative values, the final set of *USIDs* will be in order.

We observe that

- Each user is able to insert points due to the modified rounding scheme
- Negative values are not within the given interval.

Effect of condition C5.

Condition C5. (see for detail chapter 8, section 8.1.3) is introduced to strictly preserve order of identifiers based on exchanged points. This order preservation property is implemented in Algorithm 8 from line 13 to line 18. Condition C5.

is still flexible in some cases and can be excluded partially or fully. It may affect the global cardinality, particularly when identifiers are to be computed, based on exchanged identifiers and users are participating without keeping any order in participation. This section deals with observing the effect of condition C5. in the outlines (section 8.1.3, page 123). For this purpose, we have not made any change in function 7 but only exclude condition C5..

Computing *LCIDs* keeping *USIDs* in ascending Order

We perform the similar experiment as above.

Example 30. *Suppose* $p_r = 1$*, user* U_1 *with* $USID = \epsilon_1 = 0.001$ *computes LCIDs for the interval* $[\,0,\, 0.2\,]$ *as given below*

$$S_1 = [\,0,\, 0.099,\, 0.199,\, 0.2\,]$$

Then U_1 *sends LCIDs to the next user* U_2 *with* $USID = \epsilon_2 = 0.002$*. User* U_2 *computes one point based on the points sent by* U_1

$$S_2 = [\,0,\, 0.098,\, 0.099,\, 0.198,\, 0.199,\, 0.2\,]$$

and sends to next users.
Now, user U_3 *with* $USID = \epsilon_3 = 0.003$ *compute point because the* ∇ *Condition is valid. For instance, for two points* $[\,0.099,\ 0.198\,]$

$$\nabla = \left| \frac{\left|0.198\right|_1 - \left|0.099\right|_1}{2} \right|_1 = 0.1 > USID_{max}$$

After the completion of process, final set of USIDs is listed in the Table 9.9

A final set after the participation of each user				
0	0.0910	0.0920	0.0930	0.0940
0.0950	0.0960	0.0970	0.0980	0.0990
0.191	0.192	0.193	0.194	0.195
0.196	0.197	0.198	0.199	0.2

Table 9.9: Set of Identifiers

Similarly, for the interval $[\,0,\ 1\,]$, and a set of nine users/sites assigned with a set of *USIDs* as given below

$$[0.001,\, 0.002,\, 0.003,\, 0.004,\, 0.005,\, 0.006,\, 0.007,\, 0.008,\, 0.009]$$

User with $USID = 0.009$ will compute 10 points, and all other users also computes 10 points. So the global cardinality for the given interval is 92 as shown in the Table 9.10. We observe that

- All users are able to compute the points.
- Final set of *LCIDs* is an ordered set.

A final set after the participation of each user									
0	0.091	0.092	0.093	0.094	0.095	0.096	0.097	0.098	0.099
0.191	0.192	0.193	0.194	0.195	0.196	0.197	0.198	0.199	
0.291	0.292	0.293	0.294	0.295	0.296	0.297	0.298	0.299	
0.391	0.392	0.393	0.394	0.395	0.396	0.397	0.398	0.399	
0.491	0.492	0.493	0.494	0.495	0.496	0.497	0.498	0.499	
0.591	0.592	0.593	0.594	0.595	0.596	0.597	0.598	0.599	
0.691	0.692	0.693	0.694	0.695	0.696	0.697	0.698	0.699	
0.791	0.792	0.793	0.794	0.795	0.796	0.797	0.798	0.799	
0.891	0.892	0.893	0.894	0.895	0.896	0.897	0.898	0.899	
0.991	0.992	0.993	0.994	0.995	0.996	0.997	0.998	0.999	1

Table 9.10: Set of Identifiers

Computing *LCIDs* keeping *USIDs* in descending Order

Consider the following example

Example 31. *Suppose* $p_r = 1$*, user* U_9 *with* $USID = \epsilon_9 = 0.009$ *computes points for the interval* $[\,0,\, 0.2\,]$*,*

$$[\,0,\, 0.091,\, 0.191,\, 0.2\,]$$

and sends to the next user U_8 *with* $USID = \epsilon_8 = 0.008$*. User* U_8 *computes one point based on the points sent by* U_9

$$[\,0,\, 0.092,\, 0.091,\, 0.192,\, 0.191,\, 0.2\,]$$

and sends LCIDs to next users.
Now, user U_7 *with* $USID = \epsilon_7 = 0.007$ *computes point because the* ∇ *Condition is valid. For instance, for two points* 0.091 *and* 0.192

$$\nabla = \left| \frac{\left|0.192\right|_1 - \left|0.091\right|_1}{2} \right|_1 = 0.1 > USID_{max}$$

User U_6 *compute points*

$$[\,0,\, 0.093,\, 0.092,\, 0.091,\, 0.193,\, 0.192,\, 0.191,\, 0.2\,]$$

Final set of USIDs computed is listed in the Table 9.11.

Thus, for an interval $I = [\,0,\ 1\,]$, and a set of nine users/sites with a set of *USIDs* as given below

$$[\,0.001,\, 0.002,\, 0.003,\, 0.004,\, 0.005,\, 0.006,\, 0.007,\, 0.008,\, 0.009\,]$$

User with $USID = 0.009$ computes 10 points, and each other user/site also compute 10 points. So the global cardinality for the given interval is 92 as shown in the Table 9.12. We observe that

- All users are able to compute the points.
- Each patch of *USIDs* is in decreasing order.

A final set of *USIDs*				
0	0.099	0.098	0.097	0.096
0.095	0.094	0.093	0.092	0.091
0.199	0.198	0.197	0.196	0.195
0.194	0.193	0.192	0.191	0.2

Table 9.11: Set of Identifiers

A final set after the participation of each user									
0	0.099	0.098	0.097	0.096	0.095	0.094	0.093	0.092	0.091
0.199	0.198	0.197	0.196	0.195	0.194	0.193	0.192	0.191	
0.299	0.298	0.297	0.296	0.295	0.294	0.293	0.292	0.291	
0.399	0.398	0.397	0.396	0.395	0.394	0.393	0.392	0.391	
0.499	0.498	0.497	0.496	0.495	0.494	0.493	0.492	0.491	
0.599	0.598	0.597	0.596	0.595	0.594	0.593	0.592	0.591	
0.699	0.698	0.697	0.696	0.695	0.694	0.693	0.692	0.691	
0.799	0.798	0.797	0.796	0.795	0.794	0.793	0.792	0.791	
0.899	0.898	0.897	0.896	0.895	0.894	0.893	0.892	0.891	
0.999	0.998	0.997	0.996	0.995	0.994	0.993	0.992	0.991	1

Table 9.12: Set of Identifiers

9.2.5 Assuring order preservation

Generally, a midpoint, of two points, computed by classical midpoint formula lies within these two points but it may not true under certain environment as explained in section 9.2.3, where a midpoint does not contained in the interval.

In generating identifiers, order preserving problem exist specifically during the exchange of updates and in computing identifiers based on remote points (identifiers), we have discussed this problem in detail in section 1.2.2 and section 1.2.3. To deal with this issue, we pose condition C5. (see, outlines, section 8.1.3) so that the order preservation property can be assured.

Next, we present a scenario of shared document, where three users participate in edition/modification, we explain, how the corresponding identifiers are created and order is preserved over them.

Multi-users scenario

We take a shared document marked with 'Beg' and 'End' (supposed to be mapped on an interval [0, 1]) as shown in Figure 9.1. Let with precision $p_r = 1$, three users U_1 (assigned a user identifier 0.007), U_2 (assigned a user identifier 0.004) and U_3 (assigned a user identifier 0.001) start editing the

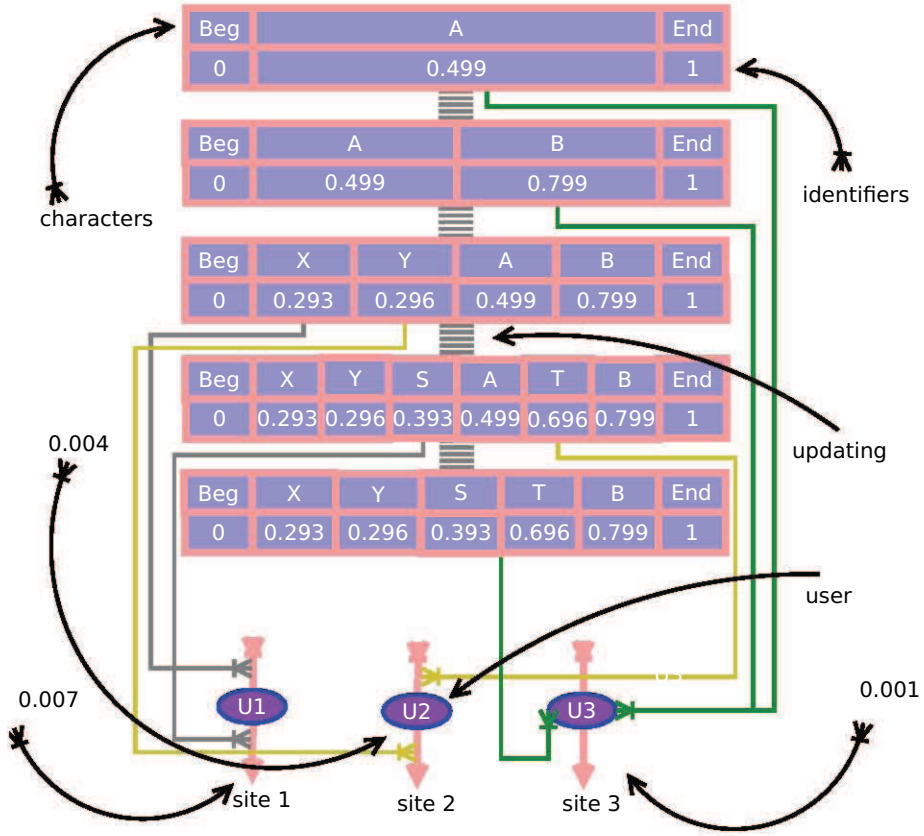


Figure 9.1: A scenario of three users

shared document at three different sites as shown in the Figure 9.1. Let user U_3 inserts character 'A' then the first associated identifier 0.499 is computed. User U_3 inserts a second character 'B' between 'A' and 'End' and the associated identifier between 0.499 and 1 is computed as 0.799. Now document is updated and at each site sequence of characters 'AB' is appeared. Now, user U_1 and user U_2 concurrently inserted two characters 'X' and 'Y' between 'Beg' and 'A'. Notice that identifiers for both characters are to be computed between 0 and 0.499 and are 0.293 and 0.296 respectively. Again, updates are performed and each user has sequence of characters 'XYAB'. Now, user U_1 inserts character 'S' between 'Y' and 'A', identifier for 'S', 0.393 is computed using neighboring identifiers of characters 'Y' and 'A'. Similar insertion of character 'T' between characters 'A' and 'B' by user U_2 is attempted. Identifier for 'T', 0.696 is computed using neighboring identifiers of characters 'A' and 'B'. At the same time user U_3 removes character 'A'. Updates are executed and each site has a sequence of characters 'XYSTB'. Notice that removal of 'A' does not affect the insertion by other users because each site has its own replicas.

Chapter 10

Exchange of points and variation in global cardinality

Contents

10.1 Delayed exchange of points 150

10.1.1 Random Participation 150

10.1.2 Ordered Participation 154

10.2 Real time exchange of points 158

10.2.1 Computing Instant Global Cardinality 159

10.2.2 Comparison 160

In decentralized collaborative editing, to generate new identifiers based on remote points and updates, the information about identifiers of remote points is also required along with updates. This will facilitate in resolving conflicts during edition/modification. We are interested in designing a model for a network of users/sites/peers in such a way that the global cardinality could be maximized.

In this chapter, we present different models to observe the effect on global cardinality during the exchange of points across the network. We provide a comparison of these models and explain the limitations of certain situations. We perform whole of experimentations with focusing in two different situations:

1. Delayed exchange of points
2. Real time exchange of points

Section 10.1 deals with the first situation whereas section 10.2 deals with the second situation in detail. For whole of the communication in the network, the identifiers are generated using the criteria (as explained in section 8.1.3, Outlines, page 123) respecting the posed precision conditions.

10.1 Delayed exchange of points

We propose this environment based on the supposition that users edit/modify shared document one after the other, for example, one of the users U_1 (*say*) starts modifying/editing shared object/document and sends updates to all other users (say lazy), then a second (any one else) user starts computing, after receiving the points from U_1. After receiving updates, any user who wants to compute points must have to verify the validity of the required conditions (section 8.1.3, Outlines, chapter 8) on the points including the points that he/she has received. This environment may prevent global cardinality in approaching to its upper bound. We explain this environment by considering two different situations, firstly when users participate randomly and secondly when they participate in an order. We describe the first situation by a stochastic-like model and second by a deterministic-like model.

10.1.1 Random Participation

This section deals with random participations of participants and it allows users to participate several times independent of any order.

Model 1

We propose a model that deals with exchange of points in the network $\aleph$, that is based on the following assumptions.

1. Each user in the network is assigned an identifier from an ordered set $\mathbb{U}$ of user/site-identifiers (*USIDs*) generated under a chosen precision.

2. Any user participate in edition/modification of shared document randomly.

3. Users perform all possible operations and send these updates, at once, to others, *i.e.*, packets of points are exchanged during the process.

4. All updates receive at all sites.

5. To generate new identifiers, it has to verify the validity of the required conditions (section 8.1.3, Outlines, chapter 8) on the points including the points received.

6. If a new identifier is generated then global cardinality will be updated otherwise it remains unchanged.

7. Any user can participate more than once, however, insertion of points depend on the validity of the required conditions.

We transform Model 1 in the form of an algorithm 10 and perform an experiment in which 90 users participate in editing a shared documents and exchange points across the network ℵ. Algorithm generates an ordered set of user identifiers and take any identifier randomly from the set. Then it computes all possible identifiers (under selected precision) for a user and send the set of these identifiers to all other users. Then algorithm take a second identifier randomly from the set that has already generated and repeat the same process as it completed first time. In fact, after first iteration, algorithm verify conditions over the identifiers including identifiers that received from previous iteration. It may create limitations in generating new identifiers due to the reason of validity of conditions of the basic criteria. Algorithm 10 provides a quick view of the Model 1 and helps to understand idea in a more simple way.

Algorithm 10: How to exchange points in Batch Mode

Algorithm: Batch Mode (*Random Participation*)
Input: Interval $I = [a_1, b_1]$, Desired precision
Output: Set of *USIDs* after exchanges and global cardinality

```
1  begin
2      Let dp := Desired precision
3          𝕌 := Set of USIDs
4          𝕃 := Set of LCIDs
5          𝕌 ⟵ GenerateUSIDs(dp)
6          USID_max:= Max. value of USID in the set 𝕌
7                  M:= No. of elements in the set 𝕌
8      for j ⟵ 1 to M do
9          USID_rand ⟵ USID randomly picked from set 𝕌
10                 𝕃 ⟵ Generate USIDs (𝕃, USID_rand,
11                                     USID_max, dp)
12         foreach USID_rand do
13             Update Global Cardinality if
14             new points are computed
15         end
16     end
17     return Final Set of LCIDs along-with global cardinality
18 end
```

Experiment

By taking rounding precision $p_r = 2$, we perform an experiment for a set of 90 users assigned with different *USIDs* and record global cardinality (corresponding to user-identifiers that participate in the experiment) in Table 10.1.

Table 10.1: Cardinality observation with precision 2

An example with 90 **users, participating randomly**					
user-identifier	0.0001900	0.0004900	0.0007500	0.0009600	0.0002500
cardinality	102	202	302	402	402
user-identifier	0.0005200	0.0006200	0.0007000	0.0005600	0.0003700
cardinality	402	402	402	402	402
user-identifier	0.0008400	0.0001200	0.0001400	0.0002000	0.0004600
cardinality	402	402	402	402	402
user-identifier	0.0008400	0.0001300	0.0003100	0.0004900	0.0006700
cardinality	402	402	402	402	402
user-identifier	0.0002000	0.0003900	0.0001500	0.0004100	0.0004900
cardinality	402	402	402	402	402
user-identifier	0.0003300	0.0008900	0.0002000	0.0003200	0.0005000
cardinality	402	402	402	402	402
user-identifier	0.0006100	0.0006700	0.0007600	0.0009000	0.0007400
cardinality	402	402	402	402	402
user-identifier	0.0007800	0.0001100	0.0004500	0.0007000	0.0009300
cardinality	402	402	402	402	402
user-identifier	0.0004000	0.0009000	0.0004000	0.0006300	0.0007600
cardinality	402	402	402	402	402
user-identifier	0.0006800	0.0007500	0.0002100	0.0005800	0.0009900
cardinality	402	402	402	402	502
user-identifier	0.0004400	0.0002400	0.0003500	0.0003300	0.0001700
cardinality	502	502	502	502	502
user-identifier	0.0007200	0.0008700	0.0003200	0.0008200	0.0003100
cardinality	502	502	502	502	502
user-identifier	0.0004100	0.0001800	0.0006200	0.0001200	0.0007800
cardinality	502	502	502	502	502
user-identifier	0.0001200	0.0008200	0.0009700	0.0004600	0.0006900
cardinality	502	502	502	502	502
user-identifier	0.0006100	0.0002500	0.0003800	0.0006000	0.0001200
cardinality	502	502	502	502	502
user-identifier	0.0005400	0.0007600	0.0004900	0.0008000	0.0008300
cardinality	502	502	502	502	502
user-identifier	0.0005800	0.0006900	0.0007800	0.0004200	0.0003900
cardinality	502	502	502	502	502
user-identifier	0.0001000	0.0009200	0.0001800	0.0007300	0.0005200
cardinality	502	502	502	502	502

We observe that two users with $USID = 0.00049$ and 0.00012 participate 4 times in the experiment and is the maximum participation, highlighted by

light-gray color. We show the jumps in global cardinality by green color and the maximum cardinality achieved during this process, by pink color. A graphical

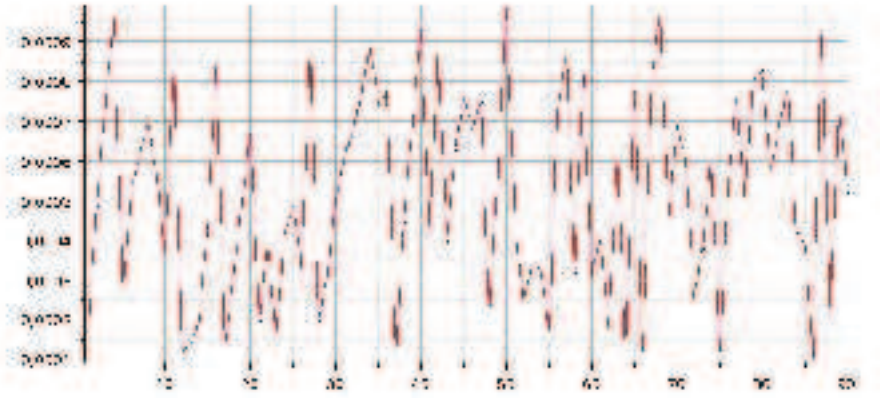


Figure 10.1: Users participation

observation is noted in Figure 10.1 that shows the participation of users in the network ℵ. X-axis presents the number of users while y-axis presents the set of *USIDs*. It is easy to observe that who participates, how many times in the experiment. There are 32 users out of 90 who did not participate in the experiment.

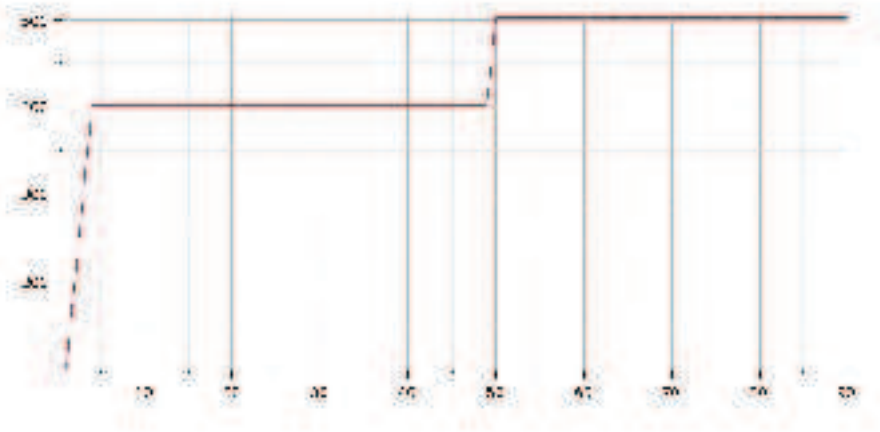


Figure 10.2: Number of users vs global cardinality

Figure 10.2 shows updates in the global cardinality corresponding to users in the network ℵ. X-axis presents the number of users while y-axis presents global cardinality corresponding to *USIDs*. It is easy to observe that which user does not make an increment in the global cardinality.

Probability of maximum global cardinality

Suppose that T_u denotes the total number of users participate in the experiment of computing points, then the probability to get maximum global cardi-

nality will be calculated by

$$\frac{1}{T_u^{T_u}}$$

Example 32. *Let the precision $p_r = 2$ and $p_\epsilon = 5$, then we have a set of 90 different users with different user-identifiers. If we perform the random experiments as one of them is shown in the table 10.1 then the probability to have the maximum global cardinality is given by*

$$\frac{1}{T_u^{T_u}} = \frac{1}{90^{90}} = 1.312726192 * 10^{-176}$$

10.1.2 Ordered Participation

In last section we discussed random participation of the participants and noted the effect on global cardinality. This section deals with the second situation in which participants participate one by one following an order.

Model 2

To deal with the second situation, we propose this model for the network $\aleph$ of users keeping following assumptions.

1. Each user in the network is assigned an identifier from an ordered set $\mathbb{U}$ of user/site-identifiers (*USIDs*) generated under a chosen precision.

2. Users participate in editing shared document keeping an order.

3. Users perform all possible operations and send these updates, at once, to others, *i.e.*, packets of points are exchanged during the process.

4. All sites in the network are able to receive the points that sent by others.

5. To generate new identifiers, it has to verify the validity of the required conditions (section 8.1.3, Outlines, chapter 8) on the points including the points received.

6. If a user computes new points then global cardinality will be updated otherwise it will remain unchanged.

7. A user can participate only once.

To notice the effect on global cardinality, we make small modification in algorithm 10 and named it algorithm 11. The modification can be seen in line 9. This small modification makes it easy to perform an experiment keeping in view Model 2.

Algorithm 11: Batch Mode Ordered Participation

Algorithm: Exchange of points

Input: Interval $I = [a_1, b_1]$, Desired precision

Output: Set of *USIDs* after exchanges and global cardinality

```
1  begin
2      Let dp := Desired precision
3          𝕌 := Set of USIDs
4          𝕃 := Set of LCIDs
5          𝕌 ⟵ GenerateUSIDs(dp)
6          USID_max:= Max. value of USID in the set 𝕌
7                  M:= No. of elements in the set 𝕌
8      for j ⟵ 1 to M do
9          USID_j ⟵ Jth USID in the set 𝕌
10                 𝕃 ⟵ Generate USIDs (𝕃, USID_j,
11                                     USID_max, dp)
12         foreach USID_j do
13             Update Global Cardinality if
14             new points are computed
15         end
16     end
17     return Final Set of LCIDs along-with global cardinality
18 end
```

Experiment

Executing algorithm 11, we perform an experiment keeping rounding precision $p_r = 2$, with a set of 90 users assigned different *USIDs*. We record the global cardinality corresponding to user-identifiers in the table 10.2.

Table 10.2: Cardinality observation with precision 2

An experiment with 90 users, participating randomly					
user-identifier	0.0001000	0.0001100	0.0001200	0.0001300	0.0001400
cardinality	102	202	302	402	502
user-identifier	0.0001500	0.0001600	0.0001700	0.0001800	0.0001900
cardinality	602	702	802	902	1002
user-identifier	0.0002000	0.0002100	0.0002200	0.0002300	0.0002400
cardinality	1102	1202	1302	1402	1502
user-identifier	0.0002500	0.0002600	0.0002700	0.0002800	0.0002900
cardinality	1602	1702	1802	1902	2002
user-identifier	0.0003000	0.0003100	0.0003200	0.0003300	0.0003400
cardinality	2102	2202	2302	2402	2502

Continued on next page

Table 10.2 – continued from previous page

An experiment with 900 users, participating randomly					
user-identifier	0.0003500	0.0003600	0.0003700	0.0003800	0.0003900
cardinality	2602	2702	2802	2902	3002
user-identifier	0.0004000	0.0004100	0.0004200	0.0004300	0.0004400
cardinality	3102	3202	3302	3402	3502
user-identifier	0.0004500	0.0004600	0.0004700	0.0004800	0.0004900
cardinality	3602	3702	3802	3902	4002
user-identifier	0.0005000	0.0005100	0.0005200	0.0005300	0.0005400
cardinality	4102	4202	4302	4402	4502
user-identifier	0.0005500	0.0005600	0.0005700	0.0005800	0.0005900
cardinality	4602	4702	4802	4902	5002
user-identifier	0.0006000	0.0006100	0.0006200	0.0006300	0.0006400
cardinality	5102	5202	5302	5402	5502
user-identifier	0.0006500	0.0006600	0.0006700	0.0006800	0.0006900
cardinality	5602	5702	5802	5902	6002
user-identifier	0.0007000	0.0007100	0.0007200	0.0007300	0.0007400
cardinality	6102	6202	6302	6402	6502
user-identifier	0.0007500	0.0007600	0.0007700	0.0007800	0.0007900
cardinality	6602	6702	6802	6902	7002
user-identifier	0.0008000	0.0008100	0.0008200	0.0008300	0.0008400
cardinality	7102	7202	7302	7402	7502
user-identifier	0.0008500	0.0008600	0.0008700	0.0008800	0.0008900
cardinality	7602	7702	7802	7902	8002
user-identifier	0.0009000	0.0009100	0.0009200	0.0009300	0.0009400
cardinality	8102	8202	8302	8402	8502
user-identifier	0.0009500	0.0009600	0.0009700	0.0009800	0.0009900
cardinality	8602	8702	8802	8902	9002

For the above table 10.2, we can compute global cardinality for any user by knowing its $USID$ and the local cardinality using the following formula.

$$lc + 2 + 100((USID) - 0.0001) * 10^5$$

where, lc denotes the local cardinality.
As an example, for $USID = 0.00088$, the global cardinality is computed as

$$102 + 100 * (0.00088 - 0.0001) * 10^5 = 7902$$

Figure 10.3 shows a contribution of each user in updating the global cardinality during the experiment whereas Figure 10.4 shows participation of each user in the experiment. In both figures, x-axis presents the number of users participate in the experiment, where y-axis shows updates in cardinality and the

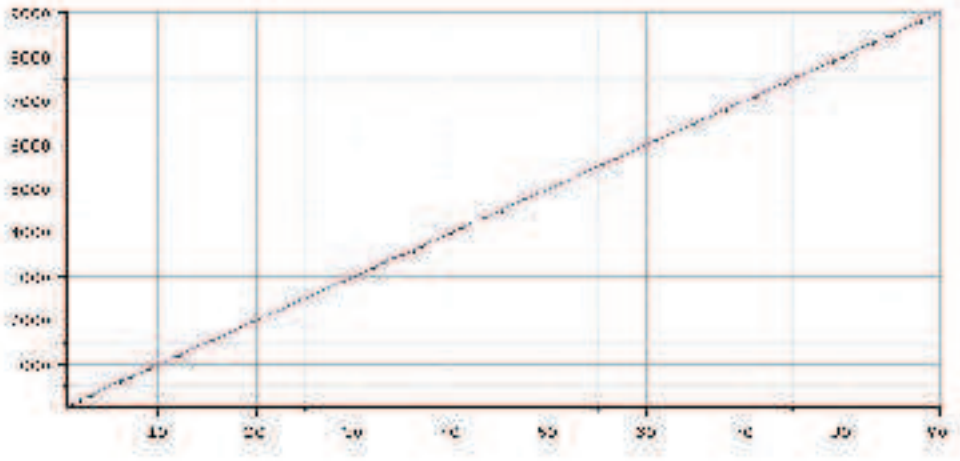


Figure 10.3: Users participation

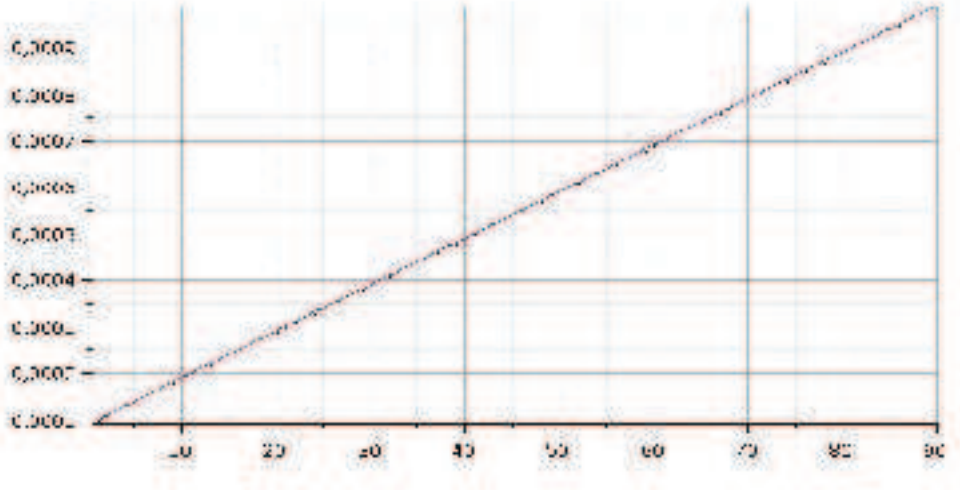


Figure 10.4: Number of users vs global cardinality

set of user-identifiers respectively. Note that both figures present straight lines showing equal participation of users in the experiment and equal contribution in updating the global cardinality.

When users participate keeping the set of *USIDs* in descending order

We keep the set of *USIDs* in decreasing order and modify condition **??** of Model 2 as

??-b. From the set $\mathbb{U}$, we pick any user/site-identifier in decreasing order, *i.e.*, next user participates in the process has $USID$ smaller than the $USID$ hold by the previous user.

Suppose that user at $USID = max(USIDs)$ starts computing points and send to all other users in the network $\aleph$ such that others are lazy and they start

computing after receiving these points. We record the global cardinality versus $USID$ in the Table 10.3.

Table 10.3: Cardinality observation with precision 1

An experiment with 9 **users**			
user-identifier	0.009	0.008	0.007
cardinality	12	12	12
user-identifier	0.006	0.005	0.004
cardinality	12	12	12
user-identifier	0.003	0.002	0.001
cardinality	12	12	12

It is clear by the Table 10.3, that only first user with $USID = 0.0090000$ succeeded to insert points but not others in the network $\aleph$ due to the reason that Condition C5. (outlines section 8.1.3, page 123) is not satisfied for other users.

Example 33. *As shown in the Table 10.3, the user at* $USID = 0.009$ *compute a set of* $LCIDs$ *for the interval* $I = [\,0,\,1\,]$ *(as listed in the Table 10.4). and*

Set of *USIDs*			
0	0.091	0.191	0.291
0.391	0.491	0.591	0.691
0.791	0.891	0.991	1

Table 10.4: Example

send to all other users. Now user/site with $USID = 0.008$ *computes the first point* 0.092 *for the sub-interval* $[\,0.0,\,0.091\,]$. *But due to Condition C5., the point* 0.092 *will not be accepted and it will leads to no update in the global cardinality.*

10.2 Real time exchange of points

Contrary to the environment discussed in section 10.1, we propose this environment based on the assumptions as illustrated in Model 3. Under this environment the identifiers are not essentially computed on the basis of remote points and all participants are supposed to be active. This environment has significant effect on global cardinality too. Real time exchange of points actually is an ideal situation to maximize the global cardinality. This situation can be modeled in the following way.

Model 3

This section deals to model the real time exchange of points (characters, identifiers). We propose a model based on the following assumptions.

1. Each user in the network is assigned an identifier from an ordered set $\mathbb{U}$ of user/site-identifiers (*USIDs*) generated under a chosen precision.
2. To edit/modify shared document, users are free to participate in a random way or in an ordered way.
3. Users perform all possible updates (and corresponding identifiers are generated following the criteria (outlines section 8.1.3, page 123)) and send these updates immediately without any delay, to others, *i.e.*, users send updates to others before receiving identifiers from others.
4. All users in the network are able to receive the points that sent by others.
5. To generate new identifiers, it is not necessary to verify the validity of the required conditions (section 8.1.3, Outlines, chapter 8) on the points received.
6. If new identifiers are generated then global cardinality will be updated otherwise it will remain unchanged.
7. A user can participate once.

Model 3 makes it possible to maximize the global cardinality. If we sort out the set of user-identifiers and corresponding cardinality in an increasing order then we get a straight line inclined with horizontal line. From the straight line, it is easy to compute global cardinality at any time.

10.2.1 Computing Instant Global Cardinality

Suppose that two users u_i and u_j at time t_i and t_j compute the global cardinality gc_i and gc_j. Suppose ue_i and ue_j are the user-identifiers assigned to U_i and U_j then a third user u_γ with ue_γ can estimate the global cardinality at time t_γ, (when u_γ completes its turn), as follows

$$gc_\gamma = \frac{gc_j - gc_i}{ue_j - ue_i}(ue_\gamma - ue_i) + gc_i$$

Due to validity of condition 3 (Model 3), each user is able to insert all the points that he/she can compute based on the computation criteria, so each user achieves his/her local cardinality. Due to the fact that each user send points immediately to all others, ultimately, each user is able to complete the process of computation before receiving points from other users, as a result, the global cardinality for this modal is the union of the local cardinalities of all users in the network and this is the upper-bound for global cardinality.

10.2.2 Comparison

This section presents a comparison of different situations that we have studied in this chapter. In Model 1, all users are not able to update the global cardinality due to the reason that Condition C5. is not valid for all users. For example, in the experiment performed under the scenario of Model 1, user with $USID = 0.00099$ succeeded to insert 100 points and updates the global cardinality from 402 to 502 but user with $USID = 0.00044$ did not succeed to insert any point and ultimately, there is no update in the global cardinality is observed.

Model 2, presents the situation of ordered participation and in the experiment performed a user with $USID = 0.0001$ starts edition/modification and sends updates to others. Since the users are assigned the $USID$ in an increasing order and they follow this order in sending the points. Therefore Criteria (outlines section 8.1.3, page 123) including Condition C5. both remains valid for each user in the network $\aleph$ and each user succeed to make edition/modification. It enables each user to achieve its local cardinality, ultimately, each user updates global cardinality.
The most worst case is described in the table 10.3, where only one user is able to insert points but all others fail to update global cardinality. In this case, criteria (Outlines, section 8.1.3, chapter 8) is satisfied for all users but Condition C5. is not satisfied.

An ideal situation in maximizing the global cardinality achieved by implementing Model 3. The global cardinality achieved both in Model 2 and Model 3 is similar. But in Model 2, the users have to follow the order imposed in computing and sending the points where as Model 3 (section 10.2) is independent of such restrictions.

Part III

Evaluation

Chapter 11

Further work and conclusion

This thesis comprises the successful efforts in advancing the capabilities of sequential computation and distributed systems through investigating, extending and introducing strategies that facilitate *"In Situ Design of Computation (IDC)"* (part-I) and indexed communications in decentralized collaborative editing systems (part-II). The key contributions aimed to focus at:

1. Sequential break-down of operations through *in situ* design of computation (IDC).

2. Designing decentralized collaborative editing system based on precision control indexing technique (PCT).

This final chapter presents future work and further perspectives of the described results.

11.1 Further work

We begin by discussing possible future research directions and identifying open questions that have emerged from this thesis.

11.1.1 Dispersion

In collaborative editing, the process of exchange of points corresponds to the insertion/deletion of points during the edition/modification of shared document. The removal of point creates the possibility to insert new point. But each removal of point does not guarantee the insertion of new point as one can desire. We describe this problem as follows.

Problem Description

Suppose that an experiment is performed in which a shared document id edited. Let, for instance, there are nine points in the document (see figure 11.1) as given below.

$$[p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9]$$

Suppose that no more points can be added. Let one of the users attempted

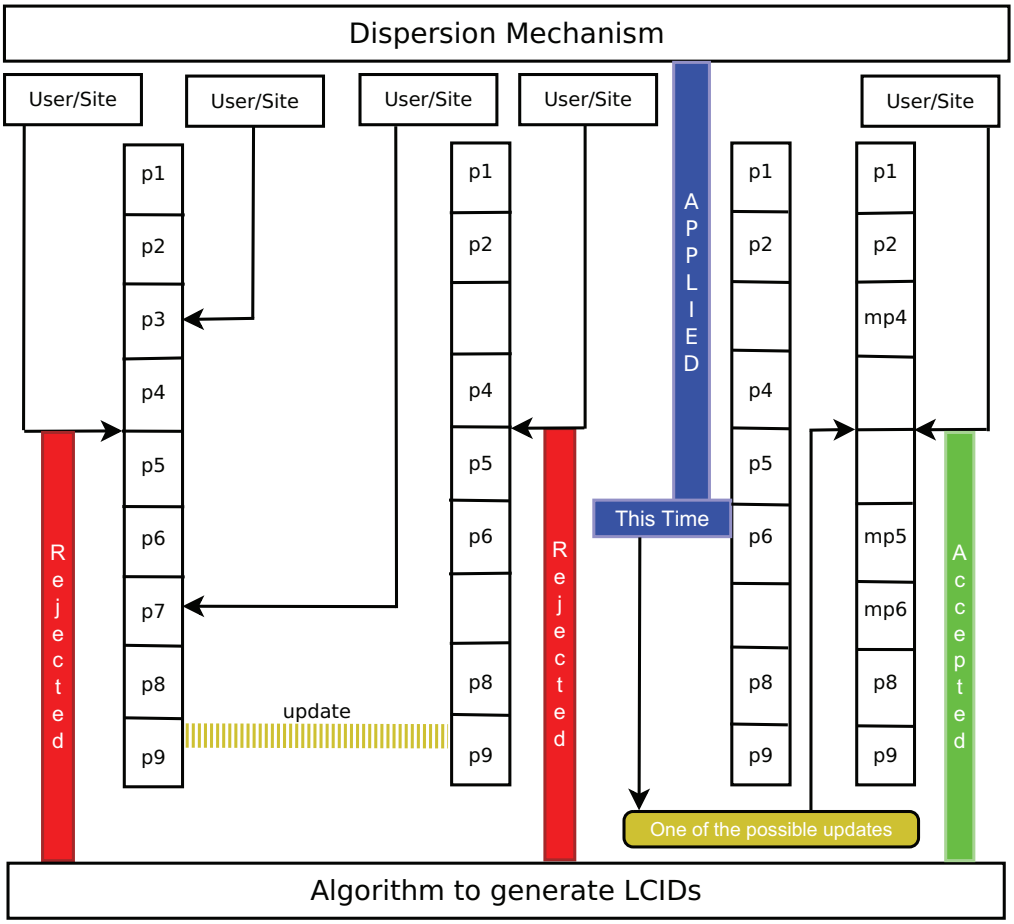


Figure 11.1: General diagram for dispersion problem

to insert a point at the position between points p_4 and p_5 and Algorithm 8 is executed to perform the task, but the request is not accepted. Next, suppose that one/two user/site/users/sites removes/remove two points p_3 and p_7 and updates are executed. A different/same user/site again attempted the insertion of a point at the position between the points p_4 and p_5. But still the request is not accepted by the Algorithm 8 because of the failure of conditions.
Suppose that there is a dispersion mechanism that can displace the points in such a way that available space/memory can be fully utilized. Now, apply the dispersion mechanism to create the space to facilitate the intension of user/site.

After the execution of updates, the points mp_4, mp_5 and mp_6 are the modified points.
Notice that, a user/site attempted the insertion of point at the same position and the request is now accepted by the Algorithm 8.
In fact, we are interested to design a comprehensive "Dispersion Mechanism", that enables to recover empty spaces (or memory slots).

Next, we explain one of the possible situations and propose one of the possible algorithms that could be helpful to understand the dispersion problem.

11.1.2 One of the possible situations

Suppose that there are two users U_1 and U_2, assigned with *USIDs*, ϵ_1 and ϵ_2 such that

$$USID_{max} = max(\epsilon_1, \epsilon_2)$$

, compute, and let two sets S_1 and S_2 of *LCIDs* are computed such that

$$S_1 \cup S_2 = S\ (say)$$

Let

$$p_j,\ 1 \leq j \leq Cd_S$$

denotes position of points in the set S, where Cd_S denotes number of elements in the set S. Let users U_1, U_2 are not allowed to remove points at positions p_j, for $j = 1$ and/or $j = Cd_S$.
Let n_r denotes number of points removed and n_{in} denotes number of points that user can insert then $n_{in} = n_r$. Now, let U_1/U_2 remove n_r consecutive points at position p_j,

$$n_1 \leq j \leq n_2$$

for some $n_1, n_2 \in \mathbb{N}$.

11.1.3 Case-1

Let U_1/U_2 wants to insert points at the same position p_j,

$$n_1 \leq j \leq n_2$$

In this case, due to the closure property, users U_1/U_2 will recompute the points that has already been removed by U_1/U_2.

11.1.4 Case-2

Let U_1/U_2 wants to insert points at the position different than p_j,

$$n_1 \leq j \leq n_2,\ n_1 > 1,\ n_2 < Cd_S$$

then users U_1/U_2 will insert points $n_{in} = n_r$ at position p_j, for

$$n_0 \leq j < n_1,\ or,\ n_2 < j \leq n_3,\ n_2 > 1,\ n_3 < Cd_S,\ n_0 > 1$$

- Let p_j, $n_0 \leq j < n_1$, $n_0 > 1$ and $n_{in} = n_r$. In this case, we have two options.

    1. We add $\delta * n_r$ to all the points at position p_j, $n_1 \leq j \leq n_2$ and push them towards right-hand to create space before inserting points n_r, where $\delta = 1 \times 10^{-p_r}$.
    2. We add δ to all the points at position p_j, $n_1 \leq j \leq n_2$ each time before inserting each point and shift them towards right-hand to create space, and perform this process recursively.

- Let p_j, $n_2 < j \leq n_3$, and $n_{in} = n_r$. In this case, we repeat any one of the above two options and subtract δ from the points at position p_j, $n_1 \leq j \leq n_2$ to shift them towards left.

We generalize this procedure in the form of an algorithm as follows:
Let Cd_{SR} denotes cardinality of the set S after removing some points.

Algorithm

1. If $Cd_{SR} = Cd_S$, then no insertion is possible.
2. If $Cd_{SR} \neq Cd_S$, then execute the Algorithm 8
    (a) If conditions of Algorithm 8 are satisfied then insert the point/points, otherwise go to next step
    (b) Let user desires to insert a point between positions p_j and p_{j+1} then locate the empty positions in both directions, to the left of the p_j and to the right of the p_{j+1}.
    Suppose that p_{j+q_1} or p_{j-q_2} is empty position for some q_1, $q_2 \in \mathbb{N}$ then
        - Add δ to point (points) at position (positions) p_i, for, $j + 1 \leq i \leq j + q_1$, OR,
        - Subtract δ to point (points) at position (positions) p_i, for, $j - q_2 \leq i \leq j$, OR,
        - One can perform both operations simultaneously.
    (c) Insert the point if it is different from s, $\forall s \in S$, where
    $$S = \bigcup S_i$$
3. Continue the process until the condition $Cd_{SR} \neq Cd_S$ holds.

11.1.5 Modifications and extensions of *IDC*

We observed limitations in the *IDC* strategies (designed to deal with sequential break down of operations) that need attention to be resolved and the strategies need some modifications to deal with such limitations.

To highlighted these limitations for the fields and provided counter example that leads towards the modifications of the existing approach. An alternate approach (using Bézout's Identity) to generate sequence of assignments that performs *in situ* computation for mappings with two dimensions has been discussed but the idea needs its further extension for general case and it requires to investigate the possibility for the case of integers.

Similar limitations are observed in the case of rings and are highlighted by providing counter example in generating sequence of assignments (from the sequence of assignments that compute given mappings) to compute inverse mappings. Thus, current approach requires modifications to deal with such situations and could be applied possibly in different way.

In case of integers, the combinatorial results and counter examples show that there are certain limitations for *in situ* strategies and it require improvements and modifications. For example, reducing the number of assignments (involved in computation), providing the possibility to generate sequence of assignments that could compute inverse mapping and extending the approach for general case. Moreover, in reducing number of assignments, we have to deal carefully with some situations, *e.g*, packing similar assignments (involved in computation) may results in an assignment with a co-efficient to be multiplied.

A tool for generating sequence of assignments for *in situ* computation of bijective boolean mappings through construction of polynomials over GF(2) is a basic attempt to develop the required strategy. But still, it remains to extend the idea for general case and to find some useful combinatorial results. Throughout part I, investigation mostly remains limited to deal with linear mappings. It is still remain to investigate for the more general case.

11.2 Conclusion

This thesis aims at developing strategies to enhance the power of sequential computation and distributed systems particularly it deals with sequential break down of operations and decentralized collaborative editing systems. Our contributions lead to design decentralized collaborative editing system based on precision control indexing method introduced in Part II. Along with evaluation of some existing techniques, it explore algorithms, applications and experimentation performed to verify the results. Such indexed communication between distributed system ensures the reduction of conflicts. Since the uniqueness of identifiers for indexing is proved, therefore, there is no chance of conflicts in ordered or random exchange of points. A prescribed pattern of generated identifiers allowed to pack them in allocated space, set or table, from where, they are easy to access. Dealing with sequential break down of operations, we explore limitations of the existing strategies, extended the idea by introducing

new strategies. These strategies lead towards optimization (processor, compiler, memory, code). This style of decomposition attracts research community for further investigation and practical implementation that could lead towards designing an arithmetic unit.

Appendix A

Experimentation Frame Work

Contents

A.1 Exchange of points (an experiment with 900 Users) 169

A.1.1 Delayed exchange of points with random participation 169

A.1.2 Delayed exchange of points with ordered participation 180

In this chapter we describe experiments performed in the context of decentralized collaborative editing system.

A.1 Exchange of points (an experiment with 900 Users)

We discuss two situations, section A.1.1 describes delayed exchange of points with random participation, whereas, section A.1.2 describes delayed exchange of points with ordered participation.

A.1.1 Delayed exchange of points with random participation

We generate an ordered set $\mathbb{U}$ of 900 unique user-identifiers, with initial value of 0.00001 and final value of 0.0000999 keeping precision $p_r = 3$ and $p_\epsilon > 3$. Our algorithm picks random identifier from the set $\mathbb{U}$ and computes cardinality, then picks another value from set $\mathbb{U}$ and compute the cardinality, if the later user adds in the cardinality then it updates the global cardinality otherwise it remains unchanged. We continue this process and record the values in Table A.1.

Table A.1: Cardinality observation with precision 3

An experiment with 900 users, participating randomly					
user-eps	0.0000109	0.0000402	0.0000290	0.0000148	0.0000782
cardinality	1002	2002	2002	2002	3002
user-eps	0.0000916	0.0000371	0.0000304	0.0000966	0.0000124
cardinality	4002	4002	4002	5002	5002
user-eps	0.0000868	0.0000515	0.0000638	0.0000678	0.0000687
cardinality	5002	5002	5002	5002	5002
user-eps	0.0000712	0.0000659	0.0000120	0.0000745	0.0000355
cardinality	5002	5002	5002	5002	5002
user-eps	0.0000503	0.0000913	0.0000871	0.0000843	0.0000439
cardinality	5002	5002	5002	5002	5002
user-eps	0.0000701	0.0000373	0.0000861	0.0000691	0.0000801
cardinality	5002	5002	5002	5002	5002
user-eps	0.0000859	0.0000163	0.0000363	0.0000680	0.0000970
cardinality	5002	5002	5002	5002	6002
user-eps	0.0000951	0.0000761	0.0000704	0.0000426	0.0000756
cardinality	6002	6002	6002	6002	6002
user-eps	0.0000441	0.0000462	0.0000499	0.0000824	0.0000207
cardinality	6002	6002	6002	6002	6002
user-eps	0.0000580	0.0000331	0.0000448	0.0000673	0.0000451
cardinality	6002	6002	6002	6002	6002
user-eps	0.0000636	0.0000805	0.0000337	0.0000545	0.0000242
cardinality	6002	6002	6002	6002	6002
user-eps	0.0000828	0.0000238	0.0000451	0.0000129	0.0000247
cardinality	6002	6002	6002	6002	6002
user-eps	0.0000154	0.0000112	0.0000883	0.0000187	0.0000314
cardinality	6002	6002	6002	6002	6002
user-eps	0.0000103	0.0000675	0.0000506	0.0000361	0.0000320
cardinality	6002	6002	6002	6002	6002
user-eps	0.0000166	0.0000406	0.0000826	0.0000461	0.0000585
cardinality	6002	6002	6002	6002	6002
user-eps	0.0000555	0.0000564	0.0000922	0.0000283	0.0000769
cardinality	6002	6002	6002	6002	6002
user-eps	0.0000523	0.0000316	0.0000817	0.0000779	0.0000767
cardinality	6002	6002	6002	6002	6002
user-eps	0.0000381	0.0000950	0.0000386	0.0000134	0.0000198
cardinality	6002	6002	6002	6002	6002
user-eps	0.0000186	0.0000698	0.0000245	0.0000326	0.0000719
cardinality	6002	6002	6002	6002	6002
user-eps	0.0000793	0.0000771	0.0000941	0.0000472	0.0000992

Continued on next page

Table A.1 – continued from previous page

An experiment with 900 users, participating randomly					
cardinality	6002	6002	6002	6002	7002
user-eps	0.0000343	0.0000678	0.0000322	0.0000935	0.0000671
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000942	0.0000409	0.0000618	0.0000372	0.0000184
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000301	0.0000556	0.0000524	0.0000133	0.0000353
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000317	0.0000813	0.0000234	0.0000270	0.0000145
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000367	0.0000450	0.0000346	0.0000732	0.0000108
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000932	0.0000790	0.0000657	0.0000373	0.0000243
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000247	0.0000453	0.0000717	0.0000853	0.0000329
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000169	0.0000514	0.0000992	0.0000661	0.0000640
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000844	0.0000292	0.0000266	0.0000751	0.0000975
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000706	0.0000657	0.0000242	0.0000600	0.0000384
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000839	0.0000413	0.0000624	0.0000650	0.0000907
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000251	0.0000537	0.0000124	0.0000357	0.0000333
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000902	0.0000467	0.0000664	0.0000903	0.0000380
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000264	0.0000758	0.0000215	0.0000927	0.0000187
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000362	0.0000775	0.0000787	0.0000951	0.0000414
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000941	0.0000360	0.0000365	0.0000485	0.0000852
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000857	0.0000326	0.0000603	0.0000486	0.0000271
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000967	0.0000134	0.0000774	0.0000107	0.0000673
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000169	0.0000917	0.0000695	0.0000400	0.0000339
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000567	0.0000515	0.0000345	0.0000744	0.0000848

Continued on next page

Table A.1 – continued from previous page

An experiment with 900 users, participating randomly					
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000560	0.0000370	0.0000517	0.0000577	0.0000804
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000895	0.0000497	0.0000215	0.0000145	0.0000314
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000477	0.0000144	0.0000225	0.0000107	0.0000684
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000413	0.0000572	0.0000612	0.0000467	0.0000325
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000735	0.0000831	0.0000159	0.0000844	0.0000487
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000419	0.0000529	0.0000541	0.0000148	0.0000603
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000932	0.0000162	0.0000516	0.0000145	0.0000584
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000572	0.0000537	0.0000952	0.0000507	0.0000157
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000153	0.0000955	0.0000549	0.0000917	0.0000782
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000431	0.0000882	0.0000217	0.0000220	0.0000111
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000213	0.0000429	0.0000800	0.0000662	0.0000579
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000294	0.0000869	0.0000881	0.0000895	0.0000285
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000434	0.0000771	0.0000434	0.0000578	0.0000174
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000101	0.0000547	0.0000390	0.0000837	0.0000628
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000801	0.0000643	0.0000333	0.0000243	0.0000130
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000110	0.0000518	0.0000956	0.0000960	0.0000536
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000538	0.0000155	0.0000906	0.0000150	0.0000503
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000621	0.0000758	0.0000843	0.0000538	0.0000614
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000563	0.0000722	0.0000223	0.0000307	0.0000763
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000621	0.0000235	0.0000493	0.0000486	0.0000428

Continued on next page

Table A.1 – continued from previous page

An experiment with 900 users, participating randomly					
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000226	0.0000793	0.0000916	0.0000168	0.0000181
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000102	0.0000485	0.0000188	0.0000676	0.0000976
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000485	0.0000785	0.0000613	0.0000901	0.0000959
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000581	0.0000329	0.0000652	0.0000559	0.0000761
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000272	0.0000853	0.0000655	0.0000675	0.0000141
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000252	0.0000635	0.0000554	0.0000207	0.0000407
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000789	0.0000161	0.0000991	0.0000706	0.0000847
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000974	0.0000539	0.0000183	0.0000758	0.0000553
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000949	0.0000563	0.0000527	0.0000213	0.0000132
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000354	0.0000516	0.0000944	0.0000269	0.0000882
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000559	0.0000677	0.0000442	0.0000882	0.0000527
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000628	0.0000845	0.0000256	0.0000989	0.0000415
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000246	0.0000293	0.0000975	0.0000705	0.0000907
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000202	0.0000361	0.0000479	0.0000782	0.0000642
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000263	0.0000269	0.0000585	0.0000679	0.0000409
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000372	0.0000356	0.0000824	0.0000989	0.0000630
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000165	0.0000131	0.0000984	0.0000413	0.0000829
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000654	0.0000142	0.0000217	0.0000609	0.0000383
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000448	0.0000131	0.0000734	0.0000190	0.0000160
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000790	0.0000356	0.0000156	0.0000782	0.0000601

Continued on next page

Table A.1 – continued from previous page

An experiment with 900 users, participating randomly					
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000411	0.0000567	0.0000437	0.0000434	0.0000640
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000852	0.0000535	0.0000866	0.0000383	0.0000129
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000884	0.0000909	0.0000449	0.0000376	0.0000736
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000159	0.0000923	0.0000988	0.0000317	0.0000276
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000603	0.0000459	0.0000495	0.0000373	0.0000833
cardinality	7002	7002	7002	7002	7002
user-eps	0.0000810	0.0000385	0.0000998	0.0000643	0.0000420
cardinality	7002	7002	8002	8002	8002
user-eps	0.0000446	0.0000112	0.0000491	0.0000292	0.0000430
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000474	0.0000521	0.0000429	0.0000242	0.0000310
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000592	0.0000821	0.0000848	0.0000636	0.0000688
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000217	0.0000759	0.0000593	0.0000997	0.0000357
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000436	0.0000317	0.0000613	0.0000905	0.0000960
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000182	0.0000904	0.0000469	0.0000695	0.0000662
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000234	0.0000310	0.0000744	0.0000582	0.0000681
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000449	0.0000819	0.0000395	0.0000185	0.0000580
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000643	0.0000178	0.0000698	0.0000289	0.0000392
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000501	0.0000581	0.0000157	0.0000702	0.0000465
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000502	0.0000560	0.0000252	0.0000394	0.0000588
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000573	0.0000657	0.0000511	0.0000838	0.0000671
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000979	0.0000753	0.0000817	0.0000788	0.0000925
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000416	0.0000983	0.0000507	0.0000662	0.0000420

Continued on next page

Table A.1 – continued from previous page

An experiment with 900 users, participating randomly					
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000470	0.0000227	0.0000922	0.0000244	0.0000345
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000709	0.0000762	0.0000355	0.0000576	0.0000930
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000333	0.0000394	0.0000168	0.0000198	0.0000369
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000838	0.0000244	0.0000161	0.0000628	0.0000246
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000450	0.0000424	0.0000670	0.0000628	0.0000234
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000637	0.0000556	0.0000846	0.0000157	0.0000686
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000376	0.0000556	0.0000648	0.0000652	0.0000766
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000885	0.0000571	0.0000649	0.0000237	0.0000987
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000331	0.0000143	0.0000924	0.0000832	0.0000687
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000105	0.0000319	0.0000714	0.0000430	0.0000445
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000440	0.0000598	0.0000840	0.0000328	0.0000213
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000759	0.0000282	0.0000736	0.0000890	0.0000584
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000337	0.0000893	0.0000500	0.0000772	0.0000416
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000866	0.0000357	0.0000535	0.0000534	0.0000480
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000665	0.0000965	0.0000203	0.0000946	0.0000356
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000726	0.0000542	0.0000201	0.0000442	0.0000510
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000897	0.0000921	0.0000414	0.0000145	0.0000105
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000390	0.0000560	0.0000938	0.0000312	0.0000394
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000188	0.0000765	0.0000976	0.0000369	0.0000643
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000132	0.0000256	0.0000557	0.0000762	0.0000208

Continued on next page

Table A.1 – continued from previous page

An experiment with 900 users, participating randomly					
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000161	0.0000913	0.0000375	0.0000177	0.0000138
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000239	0.0000894	0.0000100	0.0000568	0.0000982
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000605	0.0000965	0.0000692	0.0000234	0.0000269
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000496	0.0000934	0.0000741	0.0000472	0.0000420
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000368	0.0000507	0.0000143	0.0000960	0.0000313
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000438	0.0000422	0.0000897	0.0000447	0.0000871
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000511	0.0000715	0.0000245	0.0000841	0.0000179
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000627	0.0000496	0.0000529	0.0000436	0.0000449
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000263	0.0000702	0.0000211	0.0000325	0.0000206
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000329	0.0000723	0.0000152	0.0000308	0.0000849
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000121	0.0000468	0.0000518	0.0000665	0.0000756
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000174	0.0000471	0.0000939	0.0000614	0.0000291
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000450	0.0000502	0.0000180	0.0000220	0.0000935
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000250	0.0000418	0.0000275	0.0000294	0.0000792
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000959	0.0000240	0.0000683	0.0000606	0.0000187
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000485	0.0000691	0.0000685	0.0000725	0.0000807
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000221	0.0000737	0.0000417	0.0000913	0.0000469
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000848	0.0000329	0.0000487	0.0000945	0.0000511
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000361	0.0000817	0.0000387	0.0000255	0.0000406
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000857	0.0000853	0.0000742	0.0000453	0.0000502

Continued on next page

Table A.1 – continued from previous page

An experiment with 900 users, participating randomly					
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000429	0.0000953	0.0000918	0.0000823	0.0000256
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000605	0.0000810	0.0000653	0.0000465	0.0000104
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000277	0.0000672	0.0000493	0.0000546	0.0000656
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000642	0.0000329	0.0000920	0.0000527	0.0000187
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000329	0.0000424	0.0000729	0.0000765	0.0000397
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000204	0.0000170	0.0000849	0.0000257	0.0000662
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000693	0.0000620	0.0000750	0.0000688	0.0000892
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000454	0.0000738	0.0000344	0.0000532	0.0000403
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000382	0.0000512	0.0000206	0.0000659	0.0000667
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000711	0.0000176	0.0000206	0.0000839	0.0000967
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000702	0.0000234	0.0000179	0.0000262	0.0000375
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000754	0.0000686	0.0000732	0.0000657	0.0000560
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000734	0.0000998	0.0000212	0.0000335	0.0000227
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000231	0.0000715	0.0000697	0.0000467	0.0000915
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000324	0.0000529	0.0000382	0.0000958	0.0000746
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000301	0.0000828	0.0000612	0.0000190	0.0000147
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000872	0.0000596	0.0000994	0.0000960	0.0000708
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000676	0.0000303	0.0000406	0.0000880	0.0000393
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000930	0.0000283	0.0000218	0.0000923	0.0000684
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000436	0.0000570	0.0000612	0.0000550	0.0000998

Continued on next page

Table A.1 – continued from previous page

An experiment with 900 users, participating randomly					
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000806	0.0000833	0.0000558	0.0000354	0.0000613
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000468	0.0000259	0.0000320	0.0000550	0.0000756
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000956	0.0000734	0.0000546	0.0000196	0.0000741
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000817	0.0000781	0.0000703	0.0000812	0.0000824
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000577	0.0000754	0.0000978	0.0000212	0.0000479
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000223	0.0000195	0.0000746	0.0000585	0.0000220
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000405	0.0000664	0.0000621	0.0000486	0.0000237
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000249	0.0000643	0.0000721	0.0000831	0.0000564
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000550	0.0000833	0.0000359	0.0000921	0.0000625
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000772	0.0000190	0.0000567	0.0000320	0.0000854
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000235	0.0000187	0.0000651	0.0000516	0.0000683
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000330	0.0000855	0.0000915	0.0000471	0.0000405
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000379	0.0000290	0.0000552	0.0000779	0.0000358
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000788	0.0000986	0.0000309	0.0000861	0.0000657
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000928	0.0000286	0.0000240	0.0000360	0.0000167
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000379	0.0000692	0.0000433	0.0000358	0.0000523
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000992	0.0000172	0.0000332	0.0000500	0.0000339
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000498	0.0000417	0.0000520	0.0000334	0.0000689
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000205	0.0000402	0.0000713	0.0000248	0.0000749
cardinality	8002	8002	8002	8002	8002
user-eps	0.0000337	0.0000755	0.0000447	0.0000579	0.0000276

Continued on next page

Table A.1 – continued from previous page

An experiment with 900 users, participating randomly					
cardinality	8002	8002	8002	8002	8002

In, Figure A.1, a set of 900 users presented along x-axis and a set of corresponding 900 user-identifiers is presented along y-axis. It is noted that, the

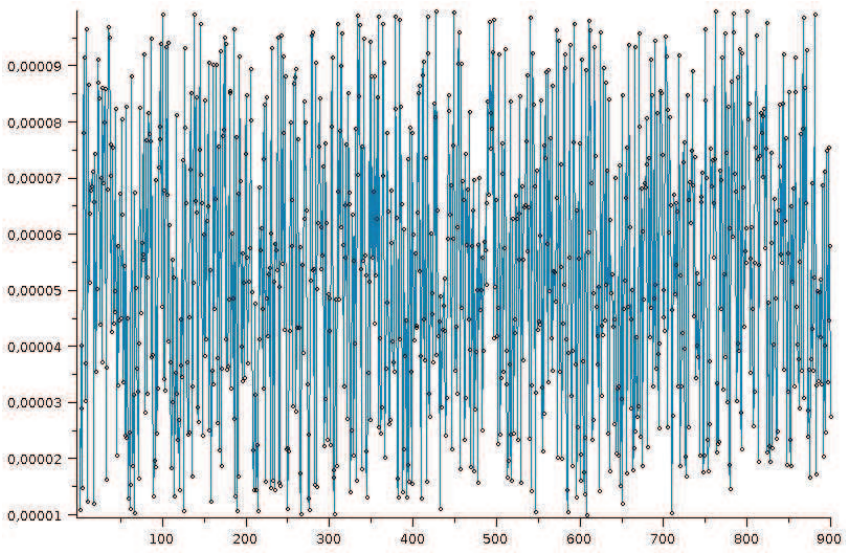


Figure A.1: A view of users participation

user with user-identifier = 0.00001, 0.00004, 0.00006 and 0.00008 participate only one time and a user with user-identifier = 0.00005 participates 2 times in the experiment, whereas users with user-identifier = 0.00002, 0.00003, 0.00007 and 0.00009 don't participate in the process. The first user who start computing the points is user with user-identifier = 0.0000109 and the last user who participates in the process is with user-identifier = 0.0000276 and it participated 2 times. The maximum value of the user-identifier participated in the experiment is 0.0000998 and this user participated 3 times. A total number of 8000 points are inserted in the interval [0, 1] (supposed to be a an empty document) during this process. Updates, at each step, in global cardinality is presented in Figure A.2.

Notice that Table A.1 and Figure A.1 shows that user with user-identifier = 0.0000329 participates 6 times in the process and it is the maximum participation of any of the 900 users, this maximum participation is highlighted by gray color in the Table A.1. We observe that 331 users participate 1 time, 173 users participate 2 times, 55 users participate 3 times, 8 users participate 4 times and there are 4 users who participate 5 times in the experiment. In total, there are 572 users who have participated in this process and out of 900,

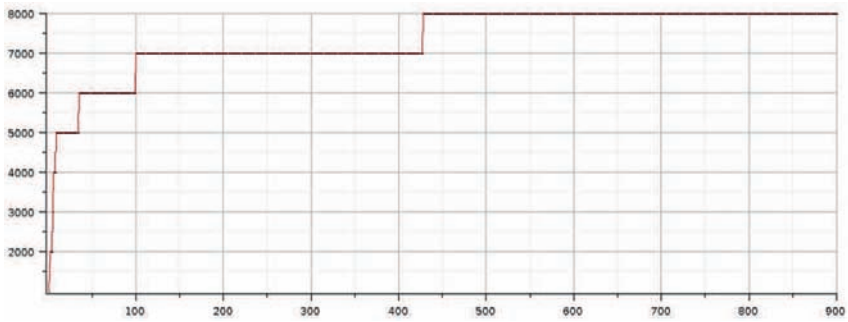


Figure A.2: Updates in global cardinality

there are 328 users who did not take part in this experiment. Green cells in the Table A.1 present an update in the global cardinality whereas pink cell shows the maximum global cardinality achieved at the completion of this experiment. A yellow cell shows the minimum and maximum value of the user-identifier participated in the experiment.

A.1.2 Delayed exchange of points with ordered participation

We design a similar algorithm as described in section A.1.1, with a modification that it does not picks random identifier from the set $\mathbb{U}$ but in an increasing order and compute cardinality, then picks next value from set $\mathbb{U}$ and compute the cardinality, if the later user adds in the cardinality then it updates the global cardinality otherwise it remains unchanged. We continue this process and record the values in the Table A.2.

Table A.2: Cardinality observation with precision 3

An experiment with 900 users, participating randomly					
user-identifier	0.0000100	0.0000200	0.0000300	0.0000400	0.0000500
cardinality	1002	2002	3002	4002	5002
user-identifier	0.0000600	0.0000700	0.0000800	0.0000900	0.0000110
cardinality	6002	7002	8002	9002	9002
user-identifier	0.0000120	0.0000130	0.0000140	0.0000150	0.0000160
cardinality	9002	9002	9002	9002	9002
user-identifier	0.0000170	0.0000180	0.0000190	0.0000210	0.0000220
cardinality	9002	9002	9002	9002	9002
user-identifier	0.0000230	0.0000240	0.0000250	0.0000260	0.0000270
cardinality	9002	9002	9002	9002	9002
user-identifier	0.0000280	0.0000290	0.0000310	0.0000320	0.0000330
cardinality	9002	9002	9002	9002	9002

Continued on next page

Table A.2 – continued from previous page

An experiment with 900 users, participating randomly					
user-identifier	0.0000340	0.0000350	0.0000360	0.0000370	0.0000380
cardinality	9002	9002	9002	9002	9002
user-identifier	0.0000390	0.0000410	0.0000420	0.0000430	0.0000440
cardinality	9002	9002	9002	9002	9002
user-identifier	0.0000450	0.0000460	0.0000470	0.0000480	0.0000490
cardinality	9002	9002	9002	9002	9002
user-identifier	0.0000510	0.0000520	0.0000530	0.0000540	0.0000550
cardinality	9002	9002	9002	9002	9002
user-identifier	0.0000560	0.0000570	0.0000580	0.0000590	0.0000610
cardinality	9002	9002	9002	9002	9002
user-identifier	0.0000620	0.0000630	0.0000640	0.0000650	0.0000660
cardinality	9002	9002	9002	9002	9002
user-identifier	0.0000670	0.0000680	0.0000690	0.0000710	0.0000720
cardinality	9002	9002	9002	9002	9002
user-identifier	0.0000730	0.0000740	0.0000750	0.0000760	0.0000770
cardinality	9002	9002	9002	9002	9002
user-identifier	0.0000780	0.0000790	0.0000810	0.0000820	0.0000830
cardinality	9002	9002	9002	9002	9002
user-identifier	0.0000840	0.0000850	0.0000860	0.0000870	0.0000880
cardinality	9002	9002	9002	9002	9002
user-identifier	0.0000890	0.0000910	0.0000920	0.0000930	0.0000940
cardinality	9002	10002	11002	12002	13002
user-identifier	0.0000950	0.0000960	0.0000970	0.0000980	0.0000990
cardinality	14002	15002	16002	17002	18002
user-identifier	0.0000101	0.0000102	0.0000103	0.0000104	0.0000105
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000106	0.0000107	0.0000108	0.0000109	0.0000111
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000112	0.0000113	0.0000114	0.0000115	0.0000116
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000117	0.0000118	0.0000119	0.0000121	0.0000122
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000123	0.0000124	0.0000125	0.0000126	0.0000127
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000128	0.0000129	0.0000131	0.0000132	0.0000133
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000134	0.0000135	0.0000136	0.0000137	0.0000138
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000139	0.0000141	0.0000142	0.0000143	0.0000144
cardinality	18002	18002	18002	18002	18002

Continued on next page

Table A.2 – continued from previous page

An experiment with 900 users, participating randomly					
user-identifier	0.0000145	0.0000146	0.0000147	0.0000148	0.0000149
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000151	0.0000152	0.0000153	0.0000154	0.0000155
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000156	0.0000157	0.0000158	0.0000159	0.0000161
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000162	0.0000163	0.0000164	0.0000165	0.0000166
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000167	0.0000168	0.0000169	0.0000171	0.0000172
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000173	0.0000174	0.0000175	0.0000176	0.0000177
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000178	0.0000179	0.0000181	0.0000182	0.0000183
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000184	0.0000185	0.0000186	0.0000187	0.0000188
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000189	0.0000191	0.0000192	0.0000193	0.0000194
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000195	0.0000196	0.0000197	0.0000198	0.0000199
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000201	0.0000202	0.0000203	0.0000204	0.0000205
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000206	0.0000207	0.0000208	0.0000209	0.0000211
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000212	0.0000213	0.0000214	0.0000215	0.0000216
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000217	0.0000218	0.0000219	0.0000221	0.0000222
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000223	0.0000224	0.0000225	0.0000226	0.0000227
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000228	0.0000229	0.0000231	0.0000232	0.0000233
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000234	0.0000235	0.0000236	0.0000237	0.0000238
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000239	0.0000241	0.0000242	0.0000243	0.0000244
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000245	0.0000246	0.0000247	0.0000248	0.0000249
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000251	0.0000252	0.0000253	0.0000254	0.0000255
cardinality	18002	18002	18002	18002	18002

Continued on next page

Table A.2 – continued from previous page

An experiment with 900 users, participating randomly					
user-identifier	0.0000256	0.0000257	0.0000258	0.0000259	0.0000261
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000262	0.0000263	0.0000264	0.0000265	0.0000266
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000267	0.0000268	0.0000269	0.0000271	0.0000272
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000273	0.0000274	0.0000275	0.0000276	0.0000277
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000278	0.0000279	0.0000281	0.0000282	0.0000283
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000284	0.0000285	0.0000286	0.0000287	0.0000288
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000289	0.0000291	0.0000292	0.0000293	0.0000294
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000295	0.0000296	0.0000297	0.0000298	0.0000299
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000301	0.0000302	0.0000303	0.0000304	0.0000305
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000306	0.0000307	0.0000308	0.0000309	0.0000311
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000312	0.0000313	0.0000314	0.0000315	0.0000316
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000317	0.0000318	0.0000319	0.0000321	0.0000322
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000323	0.0000324	0.0000325	0.0000326	0.0000327
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000328	0.0000329	0.0000331	0.0000332	0.0000333
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000334	0.0000335	0.0000336	0.0000337	0.0000338
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000339	0.0000341	0.0000342	0.0000343	0.0000344
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000345	0.0000346	0.0000347	0.0000348	0.0000349
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000351	0.0000352	0.0000353	0.0000354	0.0000355
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000356	0.0000357	0.0000358	0.0000359	0.0000361
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000362	0.0000363	0.0000364	0.0000365	0.0000366
cardinality	18002	18002	18002	18002	18002

Continued on next page

Table A.2 – continued from previous page

An experiment with 900 users, participating randomly					
user-identifier	0.0000367	0.0000368	0.0000369	0.0000371	0.0000372
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000373	0.0000374	0.0000375	0.0000376	0.0000377
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000378	0.0000379	0.0000381	0.0000382	0.0000383
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000384	0.0000385	0.0000386	0.0000387	0.0000388
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000389	0.0000391	0.0000392	0.0000393	0.0000394
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000395	0.0000396	0.0000397	0.0000398	0.0000399
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000401	0.0000402	0.0000403	0.0000404	0.0000405
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000406	0.0000407	0.0000408	0.0000409	0.0000411
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000412	0.0000413	0.0000414	0.0000415	0.0000416
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000417	0.0000418	0.0000419	0.0000421	0.0000422
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000423	0.0000424	0.0000425	0.0000426	0.0000427
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000428	0.0000429	0.0000431	0.0000432	0.0000433
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000434	0.0000435	0.0000436	0.0000437	0.0000438
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000439	0.0000441	0.0000442	0.0000443	0.0000444
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000445	0.0000446	0.0000447	0.0000448	0.0000449
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000451	0.0000452	0.0000453	0.0000454	0.0000455
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000456	0.0000457	0.0000458	0.0000459	0.0000461
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000462	0.0000463	0.0000464	0.0000465	0.0000466
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000467	0.0000468	0.0000469	0.0000471	0.0000472
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000473	0.0000474	0.0000475	0.0000476	0.0000477
cardinality	18002	18002	18002	18002	18002

Continued on next page

Table A.2 – continued from previous page

An experiment with 900 users, participating randomly					
user-identifier	0.0000478	0.0000479	0.0000481	0.0000482	0.0000483
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000484	0.0000485	0.0000486	0.0000487	0.0000488
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000489	0.0000491	0.0000492	0.0000493	0.0000494
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000495	0.0000496	0.0000497	0.0000498	0.0000499
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000501	0.0000502	0.0000503	0.0000504	0.0000505
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000506	0.0000507	0.0000508	0.0000509	0.0000511
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000512	0.0000513	0.0000514	0.0000515	0.0000516
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000517	0.0000518	0.0000519	0.0000521	0.0000522
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000523	0.0000524	0.0000525	0.0000526	0.0000527
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000528	0.0000529	0.0000531	0.0000532	0.0000533
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000534	0.0000535	0.0000536	0.0000537	0.0000538
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000539	0.0000541	0.0000542	0.0000543	0.0000544
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000545	0.0000546	0.0000547	0.0000548	0.0000549
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000551	0.0000552	0.0000553	0.0000554	0.0000555
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000556	0.0000557	0.0000558	0.0000559	0.0000561
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000562	0.0000563	0.0000564	0.0000565	0.0000566
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000567	0.0000568	0.0000569	0.0000571	0.0000572
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000573	0.0000574	0.0000575	0.0000576	0.0000577
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000578	0.0000579	0.0000581	0.0000582	0.0000583
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000584	0.0000585	0.0000586	0.0000587	0.0000588
cardinality	18002	18002	18002	18002	18002

Continued on next page

Table A.2 – continued from previous page

An experiment with 900 users, participating randomly					
user-identifier	0.0000589	0.0000591	0.0000592	0.0000593	0.0000594
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000595	0.0000596	0.0000597	0.0000598	0.0000599
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000601	0.0000602	0.0000603	0.0000604	0.0000605
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000606	0.0000607	0.0000608	0.0000609	0.0000611
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000612	0.0000613	0.0000614	0.0000615	0.0000616
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000617	0.0000618	0.0000619	0.0000621	0.0000622
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000623	0.0000624	0.0000625	0.0000626	0.0000627
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000628	0.0000629	0.0000631	0.0000632	0.0000633
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000634	0.0000635	0.0000636	0.0000637	0.0000638
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000639	0.0000641	0.0000642	0.0000643	0.0000644
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000645	0.0000646	0.0000647	0.0000648	0.0000649
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000651	0.0000652	0.0000653	0.0000654	0.0000655
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000656	0.0000657	0.0000658	0.0000659	0.0000661
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000662	0.0000663	0.0000664	0.0000665	0.0000666
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000667	0.0000668	0.0000669	0.0000671	0.0000672
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000673	0.0000674	0.0000675	0.0000676	0.0000677
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000678	0.0000679	0.0000681	0.0000682	0.0000683
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000684	0.0000685	0.0000686	0.0000687	0.0000688
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000689	0.0000691	0.0000692	0.0000693	0.0000694
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000695	0.0000696	0.0000697	0.0000698	0.0000699
cardinality	18002	18002	18002	18002	18002

Continued on next page

Table A.2 – continued from previous page

An experiment with 900 users, participating randomly					
user-identifier	0.0000701	0.0000702	0.0000703	0.0000704	0.0000705
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000706	0.0000707	0.0000708	0.0000709	0.0000711
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000712	0.0000713	0.0000714	0.0000715	0.0000716
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000717	0.0000718	0.0000719	0.0000721	0.0000722
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000723	0.0000724	0.0000725	0.0000726	0.0000727
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000728	0.0000729	0.0000731	0.0000732	0.0000733
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000734	0.0000735	0.0000736	0.0000737	0.0000738
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000739	0.0000741	0.0000742	0.0000743	0.0000744
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000745	0.0000746	0.0000747	0.0000748	0.0000749
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000751	0.0000752	0.0000753	0.0000754	0.0000755
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000756	0.0000757	0.0000758	0.0000759	0.0000761
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000762	0.0000763	0.0000764	0.0000765	0.0000766
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000767	0.0000768	0.0000769	0.0000771	0.0000772
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000773	0.0000774	0.0000775	0.0000776	0.0000777
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000778	0.0000779	0.0000781	0.0000782	0.0000783
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000784	0.0000785	0.0000786	0.0000787	0.0000788
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000789	0.0000791	0.0000792	0.0000793	0.0000794
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000795	0.0000796	0.0000797	0.0000798	0.0000799
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000801	0.0000802	0.0000803	0.0000804	0.0000805
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000806	0.0000807	0.0000808	0.0000809	0.0000811
cardinality	18002	18002	18002	18002	18002

Continued on next page

Table A.2 – continued from previous page

An experiment with 900 users, participating randomly					
user-identifier	0.0000812	0.0000813	0.0000814	0.0000815	0.0000816
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000817	0.0000818	0.0000819	0.0000821	0.0000822
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000823	0.0000824	0.0000825	0.0000826	0.0000827
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000828	0.0000829	0.0000831	0.0000832	0.0000833
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000834	0.0000835	0.0000836	0.0000837	0.0000838
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000839	0.0000841	0.0000842	0.0000843	0.0000844
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000845	0.0000846	0.0000847	0.0000848	0.0000849
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000851	0.0000852	0.0000853	0.0000854	0.0000855
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000856	0.0000857	0.0000858	0.0000859	0.0000861
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000862	0.0000863	0.0000864	0.0000865	0.0000866
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000867	0.0000868	0.0000869	0.0000871	0.0000872
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000873	0.0000874	0.0000875	0.0000876	0.0000877
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000878	0.0000879	0.0000881	0.0000882	0.0000883
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000884	0.0000885	0.0000886	0.0000887	0.0000888
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000889	0.0000891	0.0000892	0.0000893	0.0000894
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000895	0.0000896	0.0000897	0.0000898	0.0000899
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000901	0.0000902	0.0000903	0.0000904	0.0000905
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000906	0.0000907	0.0000908	0.0000909	0.0000911
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000912	0.0000913	0.0000914	0.0000915	0.0000916
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000917	0.0000918	0.0000919	0.0000921	0.0000922
cardinality	18002	18002	18002	18002	18002

Continued on next page

Table A.2 – continued from previous page

An experiment with 900 users, participating randomly					
user-identifier	0.0000923	0.0000924	0.0000925	0.0000926	0.0000927
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000928	0.0000929	0.0000931	0.0000932	0.0000933
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000934	0.0000935	0.0000936	0.0000937	0.0000938
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000939	0.0000941	0.0000942	0.0000943	0.0000944
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000945	0.0000946	0.0000947	0.0000948	0.0000949
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000951	0.0000952	0.0000953	0.0000954	0.0000955
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000956	0.0000957	0.0000958	0.0000959	0.0000961
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000962	0.0000963	0.0000964	0.0000965	0.0000966
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000967	0.0000968	0.0000969	0.0000971	0.0000972
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000973	0.0000974	0.0000975	0.0000976	0.0000977
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000978	0.0000979	0.0000981	0.0000982	0.0000983
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000984	0.0000985	0.0000986	0.0000987	0.0000988
cardinality	18002	18002	18002	18002	18002
user-identifier	0.0000989	0.0000991	0.0000992	0.0000993	0.0000994
cardinality	18002	19002	20002	21002	22002
user-identifier	0.0000995	0.0000996	0.0000997	0.0000998	0.0000999
cardinality	23002	24002	25002	26002	27002

Figure A.3 shows that a total number of 27000 points are inserted in the interval [0, 1] (supposed to be a an empty document) during this process. The straight horizontal line shows that there is no update in the global cardinality. In, Figure A.4, a set of 900 users presented along x-axis and a set of corresponding 900 user-identifiers is presented along y-axis. We have also observed that if we perform the experiment A.1.2 by keeping set $\mathbb{U}$ in decreasing order *i.e.* the user at highest value of user-identifier starts computing the points, then the global cardinality is only 1000 points. It is observed by the Table A.2 and Figure A.4 that each user in the network participates only one time. Green cells in the Table A.2 present an update in the global cardinality while the pink cell shows the maximum global cardinality achieved during this pro-

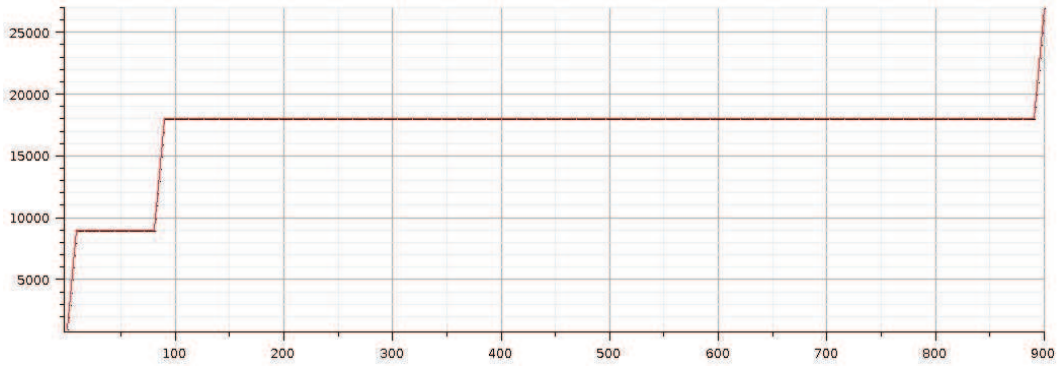


Figure A.3: Updates in global cardinality

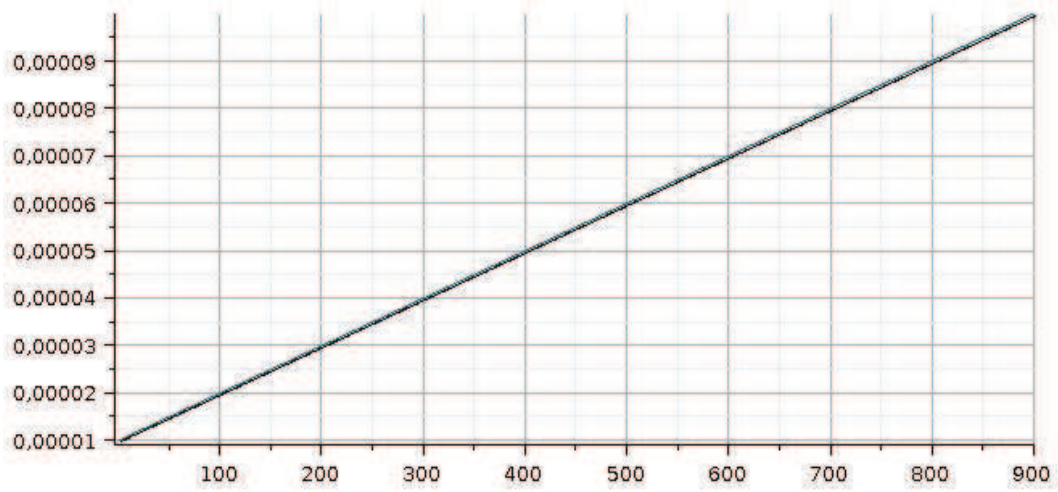


Figure A.4: A users participation curve

cess. A yellow cell shows the minimum value of the user-identifier participated in the process.

A comparison of the experiment A.1.1 and A.1.2

Comparing two experiments A.1.1 and A.1.2, a significant increment in the cardinality has been observed in the experiment A.1.2 as compared to the experiment A.1.1, but still there are 873 users out of 900 who did not insert points during the process. Since the global cardinality increased in experiment A.1.2 more than 3 times of the experiment A.1.2, therefore, clearly experiment A.1.2 is better than experiment A.1.1.

Appendix B

Implementation Framework

Contents

B.1 Algorithm 191

Mostly, we implemented our ideas using Maple and performed an extensive experimentations that, sometimes, took a number of days but here we provide one of the algorithms that implements the idea of *in situ* computation over rings.

B.1 Algorithm

```
  with(LinearAlgebra):
gcd_multipliers := proc(x::list, alpha::integer, i::integer)
local tmp , fac, projections,l,j,k,mres:
fac := ifactors(alpha)[2]:
projections := []:

for k from 1 to nops(fac) do

         l := [seq(0, j=1..nops(x))]:
      l[i] := 1:

    if igcd(x[i], fac[k][1]) <> 1 then

        for j from 1 to nops(x) do

             if igcd(x[j], fac[k][1]) = 1 then

                  l[j] := 1:
                   break:
             fi:
```

```
         od:
    fi:
       projections := [op(projections), l]:
 od:

 tmp := [seq(fac[k,1]^fac[k,2], k=1..nops(fac))]:
mres := []:
mres :=[op(mres), chrem(projections, tmp)][]:

end:
isone := proc(M::Matrix)
local i, j:
for i from 1 to RowDimension(M) do
   for j from 1 to ColumnDimension(M) do
       if M[i,j] != 1 then
         return false:
       fi:
   od:
od:
return true:
end:
creatematrix := proc(d::integer)
local M, i:
M := Matrix(d,d,0):
for i from 1 to d do
    M[i,i] := 1:
od:
return M:
end:
with(ListTools):
matrixop := proc(Ma::Matrix, N::integer)
local i,j,k,nr,nc, left, right,g,l,T,G,M, U, final, F:
M := copy(Ma):
left := []: right := []:
nr := RowDimension(M):
nc := ColumnDimension(M):

for k from 1 to min(nr, nc) do
    g:= igcd ( seq(M[i,k], i=1..nr) ):
       if g = 1 then
          continue:
       fi:

   if igcd( M[k,k]/g, N) <> 1 then
```

```
l := gcd_multipliers([seq(M[i,k]/g, i=1..nr)], N, k):

 T := creatematrix(nr)mod N:

       for j from 1 to nr do
           T[k,j] := l[j]:
       od:

         M := Multiply(T , M)mod N:
      left := [op(left), MatrixInverse(T)mod N]:
    fi:
           G:= creatematrix(nc):
     G[k,k] :=  M[k,k]:
          M := Multiply(M , MatrixInverse(G))mod N:
          U := creatematrix(nc):

      for j from 1 to nc do
         U[k,j] := M[k,j]:
      od:

          M := Multiply(M , MatrixInverse(U))mod N:
      right := [op(right), Multiply(U , G)mod N ]:
  od:

      final := []:
  for i from 1 to nops(left) do
      if isone(left[i]) = false then
         final := [op(final), left[i] ]:
      fi:
  od:

  for i from nops(right) to 1 by -1 do
      if isone(right[i]) = false then
         final :=[op(final), right[i]]:
      fi:

  od:

 Reverse(final):

end:
```

Glossary

CSCW : CSCW abbreviated for Computer Supported Collaborative Work

DCE : DCE stands for decentralized collaborative editing, where users are dispersed geographically.

DCE Model : A decentralized collaborative editing model based on precision control indexing method.

Dispersion Mechanism : A possible strategy that enables to utilize existing memory in such a way that maximum cardinality could be obtained.

Elements : Elements describe characters, lines, objects etc. that could be modified during the process of collaborative editing.

First Tool : A method to construct polynomial over GF(2) so that IDC can be constructed for boolean mappings.

IDC : IDC denotes In Situ Design of Computation, A sequential computation that use no extra variables other than the variables available as input.

LCID : LCID denotes identifiers that we assign to lines or characters in editing a shared document collaboratively.

Local Cardinality : The possible number of identifiers *USIDs* that could be created for a user/site under particular precision is called local cardinality and is denoted by C_l.

Notation (ℵ) : Notation ℵ used to denote a network comprises n ($n \in \mathbb{N}$) users, sites or peers,

PCT : Precision control technique that enables to create unique real identifiers.

Points : Points describe identifiers as well as elements because exchange of elements cause exchange of identifiers too.

POSID : POSID denotes position identifiers created to associate with characters (lines) in collaborative editing.

Precision : number of digits following the point of a value (rounded to decimal places/to significant digits), *e.g.*, the precision of the values 12.34600 and 12.345 is 5 and 3 respectively.

Precision (p_ϵ) : is a notation for precision taken to create identifiers USID.

RCE : Real-time Collaborative Edi-

tors enable group of users to edit simultaneously shared document from physically dispersed sites that are interconnected by computer network.

Round a value : A function that rounds value according to definition of Precision.

USID : USID denotes identifiers that we assign to users or sites participating in editing a shared document collaboratively.

Index

assignment matrix, 48

Bézout's identity, 41
bijection property, 132
boolean compatible, 91
boolean mapping, 78
bounds, 129

causality, 113
CCI, 114
CES, 117
churn, 114
closure properties, 133
collaborative editing, 6, 104
    centralized, 106
    decentralized, 4
compatible, 87
compiler, 22
conclusion, 99, 167
concurrency control, 105
consistency, 108
convergence, 113
CRDT, 6, 114
cross compiler, 22

DCE model, 111
    framework description, 113
delayed exchange, 150
dispersion, 163

Fibonacci numbers, 71

general boolean mappings, 84
global cardinality, 128

IDC, 31

limitations, 167
linear assignment, 32
linearity Property, 83
local cardinality, 128

ordered participation, 154
OT, 109
outlines, 123

PCT, 118
procedure, 86, 121
processor, 24
    microprocessor, 27
    processor optimization, 26

random participation, 150
    experiment, 151
rapid exchange, 158
redundancy, 11
rounding a value, 119

scalability, 114
simple mapping, 85
step mapping, 85

total ordering, 106

uniqueness of identifiers, 125
user identifiers, 120

valuation, 87

Bibliography

[Å10] Jonas Ådahl. Shared resource for collaborative editing over a wireless network. Master's thesis, Chalmers University of Technology, Gothenburg, Sweden, December 2010.

[ABE+97] Sarita V. Adve, Doug Burger, Rudolf Eigenmann, Alasdair Rawsthorne, Michael D. Smith, Catherine H. Gebotys, Mahmut T. Kandemir, David J. Lilja, Alok N. Choudhary, Jesse Z. Fang, and Pen-Chung Yew. Changing interaction of compiler and architecture. *Computer*, 30:51–58, December 1997.

[ACP+08] O. Azizi, J. Collins, D. Patil, Hong Wang, and M. Horowitz. Processor Performance Modeling using Symbolic Simulation. *ISPASS 2008, IEEE International Symposium on Performance Analysis of Systems and software*, pages 127–138, April 2008.

[Agr83] D.P. Agrawal. Graph theoretical analysis and design of multistage interconnection networks. *Computers, IEEE Transactions on*, C-32(7):637 –648, july 1983.

[AS78] W. Abu-Sufah. Improving the Performance of Virtual Memory Computers. PhD thesis, University of Illinois at Urbana-Champaign, Nov 1978.

[BCT94] Preston Briggs, Keith D. Cooper, and Linda Torczon. Improvements to graph coloring register allocation. *ACM Trans. Program. Lang. Syst.*, 16(3):428–455, 1994.

[BDGR06] Florent Bouchez, Alain Darte, Christophe Guillon, and Fabrice Rastello. Register Allocation: What does the NP-completeness Proof of Chaitin et al. Really Prove? In *Fifth Annual Workshop on Duplicating, Deconstructing and Debunking (WDDD2006) (at ISCA-33)*, June 2006. `http://perso.ens-lyon.fr/fabrice.rastello/Biblio_Perso/Articles/WDDD06.pdf`.

[Bei93] Richard Beigel. The polynomial method in circuit complexity. In *Structure in Complexity Theory Conference*, pages 82–95, 1993.

[BEM+92] A. E. Brown, J. P. Eckhardt, M. D. Mayo, W. A. Svarczkopf, and S. P. Gaur. Improved performance of IBM Enterprise System/9000 bipolar logic chips. *IBM J. Res. Dev.*, 36(5):829–834, 1992.

[BFJM89] Jean Claude Bermond, Jean Michel Fourneau, and Alain Jean-Marie. A graph theoretical approach to equivalence of multistage interconnection networks. *Discrete Appl. Math.*, 22:201–214, March 1989.

[BG08] Serge Burckel and E. Gioan. In situ design of register operations. In *Proceedings of ISVLSI'08*, pages 287–292, 2008.

[BGT09] Serge Burckel, Emeric Gioan, and Emmanuel Thomé. Mapping computation with no memory. In *Proceedings of the 8th International Conference on Unconventional Computation*, UC '09, pages 85–97, Berlin, Heidelberg, 2009. Springer-Verlag.

[BM00] Serge Burckel and M. Morillon. Three Generators for Minimal Writing-Space Computations. In *Theoretical Informatics and Application*, volume 34, pages 131–138, 2000.

[BM04a] Serge Burckel and M. Morillon. Quadratic Sequential Computations of Boolean Mappings. In *Theory of Computing Systems. 37(4)*, pages 519–525, 2004.

[BM04b] Serge Burckel and M. Morillon. Sequential computation of linear boolean mappings. In *Theoretical Computer Science, (314)*, pages 287–292. Springer, 2004.

[Bur96] Serge Burckel. Closed Iterative Calculus. In *Theoretical Computer Science*, pages 371–378, 1996.

[Bur07] Serge Burckel. The parallel-sequential duality : Matrices and graphs. *CoRR*, abs/0709.4397, 2007.

[CAC+81] Gregory J. Chaitin, Marc A. Auslander, Ashok K. Chandra, John Cocke, Martin E. Hopkins, and Peter W. Markstein. Register allocation via coloring. *Comput. Lang.*, 6(1):47–57, 1981.

[CD97] James Cohoon and Jack Davidson. *C++ Program Design: An Introduction to Programming and Object-Oriented Design*. Irwin McGraw-Hill, Boston, Massachusetts, 1997.

[Cen] IBM Systems Information Center. Compiler optimization techniques. http://publib.boulder.ibm.com/infocenter/systems.

[CH08] Y. Crama and Peter L. Hammer. *Boolean Functions - Theory, Algorithms, and Applications.* In preparation, December 02, 2008.

[Cha82] G. J. Chaitin. Register allocation & spilling via graph coloring. In *SIGPLAN '82: Proceedings of the 1982 SIGPLAN symposium on Compiler construction*, pages 98–105, New York, NY, USA, 1982. ACM.

[Chu36] A. Church. An unsolvable problem in elementary number theory. *American Journal of Mathematics*, 58:356–363, 1936.

[Cit07] Sandy Citro. *A Framework for Real Time Collaborative Editing in a Mobile Replicated Architectur.* PhD thesis, School of Computer Science and Information technology, RMIT University, Melbourne, Victoria, Australia, June 2007.

[CJL+99] Siddhartha Chatterjee, Vibhor V. Jain, Alvin R. Lebeck, Shyam Mundhra, and Mithuna Thottethodi. Nonlinear array layouts for hierarchical memory systems. In *Proceedings of the 1999 ACM International Conference on Supercomputing*, pages 444–453, 1999.

[CMR07] Sandy Citro, Jim McGovern, and Caspar Ryan. Conflict management for real-time collaborative editing in mobile replicated architectures. In Gillian Dobbie, editor, *Thirtieth Australasian Computer Science Conference (ACSC2007)*, volume 62 of *CRPIT*, pages 115–124, Ballarat Australia, 2007. ACS.

[com10] Comparing and merging files. available at: http://www.gnu.org/software/diffutils/manual/html_mono/diff.html, 2010.

[EG89] Clarence A. Ellis and Simon J. Gibbs. Concurrency Control in Groupware Systems. In *Proceedings of the ACM SIGMOD Conference on the Management of Data - SIGMOD'89*, pages 399–407, Portland, Oregon, USA, May 1989. ACM Press.

[EGR91] Clarence A. Ellis, Simon J. Gibbs, and Gail Rein. Groupware: some issues and experiences. *Commun. ACM*, 34:39–58, January 1991.

[Fal99] Bogdan J. Falkowski. A note on the polynomial form of boolean functions and related topics. *IEEE Trans. Comput.*, 48:860–864, August 1999.

[FLP+99] Matteo Frigo, Charles E. Leiserson, Harald Prokop, Sridhar Ramachandran, and Z W(l. Cache-oblivious algorithms (extended abstract). In *Proceedings of 40th Annual Symposium on Foundations of Computer Science*, pages 285–397. IEEE Computer Society Press, 1999.

[GA96] Lal George and Andrew W. Appel. Iterated register coalescing. *ACM Trans. Program. Lang. Syst.*, 18(3):300–324, 1996.

[Ghu] Abdulaziz Ghuloum. An Incremental Approach to Compiler Construction. In *Scheme and Functional Programming 2006*.

[GL89] G. H. Golub and C. F. Van Loan. Matrix Computations. *Johns Hopkins University Press*, 1989.

[GM94] Saul Greenberg and David Marwood. Real time groupware as a distributed system: concurrency control and its effect on the interface. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, CSCW '94, pages 207–217, New York, NY, USA, 1994. ACM.

[gM10] Torben Ægidius Mogensen. *Basics of Compiler Design*. DIKU, DK-2100 Copenhagen, DENMARK, August 2010.

[Hav07] F. Havet. *Graph colouring and applications*. Habilitation à diriger des recherches, Université de Nice-Sophia Antipolis, December 12 2007.

[HCL90] Jayant R. Haritsa, Michael J. Carey, and Miron Livny. Dynamic real-time optimistic concurrency control. Technical report, University of Wisconsin-Madison, 1990.

[HE08] K. Hoste and L. Eeckhout. Cole: Compiler optimization level exploration. In *the International Symposium on Code Generation and Optimization (CGO)*, 2008.

[HO92] Charles McLaughlin Hymes and Gary M. Olson. Unblocking brainstorming through the use of a simple group editor. In *Proceedings of the 1992 ACM conference on Computer-supported cooperative work*, CSCW '92, pages 99–106, New York, NY, USA, 1992. ACM.

[HSRT91] Jiandong Huang, John A. Stankovic, Krithi Ramamritham, and Donald F. Towsley. Experimental evaluation of real-time optimistic concurrency control schemes. In *Proceedings of the 17th International Conference on Very Large Data Bases*, VLDB '91, pages 35–46, San Francisco, CA, USA, 1991. Morgan Kaufmann Publishers Inc.

[Imi08] Abdessamad Imine. Decentralized concurrency control for real-time collaborative editors. In *Proceedings of the 8th international conference on New technologies in distributed systems*, NOTERE '08, pages 41:1–41:9, New York, NY, USA, 2008. ACM.

[Imi09] Abdessamad Imine. Coordination model for real-time collaborative editors. In John Field and Vasco Thudichum Vasconcelos, editors, *COORDINATION*, volume 5521 of *Lecture Notes in Computer Science*, pages 225–246. Springer, 2009.

[IMOR03] Abdessamad Imine, Pascal Molli, Gérald Oster, and Michaël Rusinowitch. Proving correctness of transformation functions in real-time groupware. In Kari Kuutti, Eija Helena Karsten, Geraldine Fitzpatrick, Paul Dourish, and Kjeld Schmidt, editors, *ECSCW*, pages 277–293. Springer, 2003.

[IOM+07] Claudia-Lavinia Ignat, Gérald Oster, Pascal Molli, Michèle Cart, Jean Ferrié, Anne-Marie Kermarrec, Pierre Sutra, Marc Shapiro, Lamia Benmouffok, Jean-Michel Busca, and Rachid Guerraoui. A comparison of optimistic approaches to collaborative editing of Wiki pages. In *Collaborative Comp.: Networking, Apps. and Worksharing (CollaborateCom)*, number 3, White Plains, NY, USA, November 2007.

[IR87] R. Ismail and V.M. Rooney. *Microprocessor Hardware and Software Concepts*. Macmillan, Collier Macmillan, New York, London, 1987.

[IROM06] Abdessamad Imine, Michaël Rusinowitch, Gérald Oster, and Pascal Molli. Formal design and verification of operational transformation algorithms for copies convergence. *Theor. Comput. Sci.*, 351(2):167–183, 2006.

[JBC05] Bo Jiang, Jiajun Bu, and Chun Chen. Preserving consistency in distributed embedded collaborative editing systems. In Zhaohui Wu, Chun Chen, Minyi Guo, and Jiajun Bu, editors, *Embedded Software and Systems*, volume 3605 of *Lecture Notes in Computer Science*, pages 601–606. Springer Berlin / Heidelberg, 2005.

[JDD90] S. Hammarling J. Dongarra, J. Du Croz and I. Duff. A set of level 3 basic linear algebra subprograms. *ACM Transactions on Mathematical Software*, pages 1–17, March 1990.

[JS92] H. V. Jagadish and Jagadish Oded Shmueli. A proclamation-based model for cooperating transactions. In *Proceedings of the*

eighteenth International Conference on Very Large Databases, pages 265–276, 1992.

[Jur01] Michael Jurczyk. Performance comparison of wormhole-routing priority switch architectures. In *Proceedings of the international conference on parallel and distributed processing techniques and applications 2001 (PDPTA'01)*, pages 1834–1840, Las Vegas, USA, 2001.

[kao10a] Differential synchronization overview. available at: `http://neil.fraser.name/writing/sync/`, 2010.

[kao10b] Wikipedia: a free, web-based, collaborative, multilingual encyclopedia. `http://en.wikipedia.org/wiki/Wikipedia`, 2010.

[KBL93] A. Karsenty and M. Beaudouin-Lafon. An algorithm for distributed groupware applications. In *Distributed Computing Systems, 1993., Proceedings of the 13th International Conference on*, pages 195–202, May 1993.

[Kel96] J. Keller. The 21264: a superscalar alpha processor with out-of-order execution. *9th Annual Microprocessor Forum*, 1996.

[KGS87] U. Meier K. Gallivan, W. Jrdby and A. Sameh. The impact of hierarchical memory systems on linear algebra algorithm design. Technical Report, University of Illinios, 1987.

[Kle36] S. C. Kleene. λ-definability and recursiveness. *Duke Mathematical Journal*, 2:340–353, 1936.

[Kön16] D. König. Über Graphen und ihre Anwendung auf Determinantentheorie und Mengenlehre. *Mathematische Annalen*, 77:453–465, 1916.

[KP93] Michael Knister and Atul Prakash. Issues in the design of a toolkit for supporting multiple group editors. *Computing Systems – The Journal of the Usenix Association*, 6:135–166, 1993.

[Kum96] A. Kumar. The HP PA-8000 RISC CPU: A high performance out-of-order processor. In *Hot Chips VIII*, 1996.

[Lam44] Gabriel Lamé. Note sur la limite du nombre des divisions dans la recherche du plus grand commun diviseur entre deux nombres entiers. volume 19, pages 867–870, Paris, 1844. Comptes Rendus de l'Académie des sciences.

[Lam78] Leslie Lamport. Time, Clocks, and the Ordering of events in a Distributed System. *Communications of the ACM*, 21(7):558–565, July 1978.

[Leo98] V. K. Leont'ev. Certain problems associated with boolean polynomials. *Computing Center, Russian academy of Sciences, ul. Vavilova 40, Moscow, GSP-1, 117967 Russia*, September 11, 1998.

[LHWBD06] Stefania Leone, Thomas B. Hodel-Widmer, Michael H. Böhlen, and Klaus R. Dittrich. Tendax, a collaborative database-based real-time editor system. In Yannis E. Ioannidis, Marc H. Scholl, Joachim W. Schmidt, Florian Matthes, Michael Hatzopoulos, Klemens Böhm, Alfons Kemper, Torsten Grust, and Christian Böhm, editors, *EDBT*, volume 3896 of *Lecture Notes in Computer Science*, pages 1135–1138. Springer, 2006.

[LL04a] Du Li and Rui Li. Ensuring content and intention consistency in real-time group editors. In *ICDCS*, pages 748–755. IEEE Computer Society, 2004.

[LL04b] Du Li and Rui Li. Preserving operation effects relation in group editors. In James D. Herbsleb and Gary M. Olson, editors, *CSCW*, pages 457–466. ACM, 2004.

[LL07] Rui Li and Du Li. A new operational transformation framework for real-time group editors. *IEEE Trans. Parallel Distrib. Syst.*, 18:307–319, March 2007.

[LRW91] Monica S. Lam, Edward E. Rothberg, and Michael E. Wolf. The cache performance and optimizations of blocked algorithms. In *Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 63–74, 1991.

[MC69] A. C. McKeller and E. G. Coffman. The organization of matrices and matrix operations in a paged multi-programming environment. *CACM*, 12(3):153–165, 1969.

[Mir01] John Miranda. The Usage of Compiler Optimization by Programmers. In *Thesis, School of Engineering and Applied Science University of Virginia*, 2001.

[MO56] R.C. Miller and B. J. Oldfield. Producing Computer Instructions for the PACT I Compiler. *Journal of the ACM*, 1956.

[MO92] L. McGuffin and G. Olson. Shredit: A shared electronic workspace. Technical report, Cognitive Science and Machine Intelligence Laboratory, University of Michigan, 1992.

[MPP02] Joon-Sang Park Michael, Michael Penner, and Viktor K Prasanna. Optimizing graph algorithms for improved cache

performance. In *Proc. Int'l Parallel and Distributed Processing Symp. (IPDPS 2002), Fort Lauderdale, FL*, pages 769–782, 2002.

[MS77] F. J. MacWilliams and N. J. A. Sloane. *The theory of error correcting codes / F.J. MacWilliams, N.J.A. Sloane*. North-Holland Pub. Co. ; sole distributors for the U.S.A. and Canada, Elsevier/North-Holland, Amsterdam ; New York : New York :, 1977.

[MSLW91] E. E. Rothberg M. S. Lam and M. E. Wolf. The cache performance and optimization of blocked algorithms. In *The Sixth International Conference on Architectural Support for Programming Languages and Operating Systems*, April 1991.

[Neu94] B. Clifford Neuman. Scale in distributed systems. In *Readings in Distributed Computing Systems*, pages 463–489. IEEE Computer Society Press, 1994.

[OMUI06] Gérald Oster, Pascal Molli, Pascal Urso, and Abdessamad Imine. Tombstone Transformation Functions for Ensuring Consistency in Collaborative Editing Systems. In *IEEE Conference on Collaborative Computing: Networking, Applications and Worksharing - CollaborateCom 2006 Collaborative Computing: Networking, Applications and Worksharing, 2006. CollaborateCom 2006. International Conference on*, pages 1–10, Atlanta, Georgia, USA, 11 2006. IEEE. `http://ieeexplore.ieee.org/`.

[OOSC92] Judith S. Olson, Gary M. Olson, Marianne Storrøsten, and Mark Carter. How a group-editor changes the character of a design meeting as well as its outcome. In *Proceedings of the 1992 ACM conference on Computer-supported cooperative work*, CSCW '92, pages 91–98, New York, NY, USA, 1992. ACM.

[Ost06] Gérald Oster. Tombstone transformation functions for ensuring consistency in collaborative editing systems. In *Proceedings of the 2nd International Conference on Collaborative Computing: Networking, Applications and Worksharing*. IEEE Press, 2006.

[OUMI05] Gérald Oster, Pascal Urso, Pascal Molli, and Abdessamad Imine. Proving correctness of transformation functions in collaborative editing systems. Research Report RR-5795, INRIA, 2005.

[OUMI06] Gérald Oster, Pascal Urso, Pascal Molli, and Abdessamad Imine. Data consistency for p2p collaborative editing. In

Pamela J. Hinds and David Martin, editors, *CSCW*, pages 259–268. ACM, 2006.

[PC04] Franjo Plavec and Tomasz Czajkowski. Distributed file replication system based on freepastry dht, 2004.

[Pin99] Tomasz Pinkiewicz. *Design of a 32-bit Arithmetic Unit based on Composite Arithmetic and its Implementation on a Field Programmable Gate Array*. Honours, University of Tasmania, 1999.

[PKBP00] Neungsoo Park, Dongsoo Kang, Kiran Bondalapati, and Viktor K. Prasanna. Dynamic data layouts for cache-conscious factorization of dft. In *Proc. of International Parallel and Distributed Processing Symposium*, pages 693–701, 2000.

[PMSL09] Nuno Preguiça, Joan Manuel Marquès, Marc Shapiro, and Mihai Leţia. A commutative replicated data type for cooperative editing. *Distributed Computing Systems, International Conference on*, 0:395–403, 2009.

[Pre07] Jon A. Preston. *Rethinking Consistency Management in Real-Time Collaborative Editing Systems*. PhD thesis, College of Arts and Sciences, Georgia State University, 2007.

[QC09] Wu Qinyi and Pu Calton. Consistency in real-time collaborative editing systems based on partial persistent sequences. Technical Report CERCS ; GIT-CERCS-09-07, Georgia Institute of Technology, 2009.

[RAS97] C. P. Ravikumar, R. Aggarwal, and C. Sharma. A graph-theoretic approach for register file based synthesis. In *Proceedings of the Tenth International Conference on VLSI Design: VLSI in Multimedia Applications*, VLSID '97, pages 118–, Washington, DC, USA, 1997. IEEE Computer Society.

[RNRä96] Matthias Ressel, Doris Nitsche-Ruhland, and Rul Gunzenh äuser. An integrating, transformation-oriented approach to concurrency control and undo in group editors. In *CSCW*, pages 288–297, 1996.

[RNRGa96a] Matthias Ressel, Doris Nitsche-Ruhland, and Rul Gunzenh äuser. An integrating, transformation-oriented approach to concurrency control and undo in group editors. In *Proceedings of the 1996 ACM conference on Computer supported cooperative work*, CSCW '96, pages 288–297, New York, NY, USA, 1996. ACM.

[RNRGA96b] Matthias Ressel, Doris Nitsche-Ruhland, Rul Gunzenhäuser, and Rul Gunzenh Auser. An integrating, transformation-oriented approach to concurrency control and undo in group editors. pages 288–297. ACM Press, 1996.

[Rud04] S. Rudeanu. On the decomposition of boolean functions via boolean equations. *j-jucs*, 10(9):1294–1301, 2004. `http://www.jucs.org/jucs_10_9/on_the_decomposition_of`.

[Sar08] Vivek Sarkar. Code optimization of parallel programs: evolutionary vs. revolutionary approaches. In *CGO '08: Proceedings of the sixth annual IEEE/ACM international symposium on Code generation and optimization*, pages 1–1, New York, NY, USA, 2008. ACM.

[Sar09] Vivek Sarkar. Challenges in code optimization of parallel programs. In *CC*, page 1, 2009.

[Sch73] P. B. Schneck. A Survey of Compiler Optimization Techniques. In *Proceedings of ACM Conference*, pages 106–113, 1973.

[Sch98] W.G. Schneeweiss. On the polynomial form of boolean functions: derivations and applications. *Computers, IEEE Transactions on*, 47(2):217 –221, feb 1998.

[SCT] K. S. McKinley S. Carr and C. Tseng. Compiler optimization for improving data locality. *Proceedings of the Sixth International Conference on Architectural Support for Programming Languages and Operating Systems*.

[SE98] Chengzheng Sun and Clarence Ellis. Operational transformation in real-time group editors: issues, algorithms, and achievements. In *CSCW '98: Proceedings of the 1998 ACM conference on Computer supported cooperative work*, pages 59–68, New York, NY, USA, 1998. ACM.

[SJZ+98] Chengzheng Sun, Xiaohua Jia, Yanchun Zhang, Yun Yang, and David Chen. Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems. *ACM Trans. Comput.-Hum. Interact.*, 5:63–108, March 1998.

[Sla92] Michael Slater, editor. *A guide to RISC microprocessors*. Academic Press Professional, Inc., San Diego, CA, USA, 1992.

[SM96] Chengzheng Sun and P. Maheshwari. An efficient distributed single-phase protocol for total and causal ordering of group operations. In *Proceedings of the Third International Conference on High-Performance Computing (HiPC '96)*, HIPC '96, page 295, Washington, DC, USA, 1996. IEEE Computer Society.

[SPBZ11] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. A comprehensive study of Convergent and Commutative Replicated Data Types. Technical Report 7506, January 2011. `http://hal.archives-ouvertes.fr/inria-00555588/`.

[SRAC91] Ian L. Sayers, Adrian P. Robson, Alan E. Adams, and E. Graeme Chester. *Principles of microprocessors*. CRC Press, Inc., Boca Raton, FL, USA, 1991.

[SS71] D. S. Scott and C. Strachey. Towards a mathematical semantics for computer languages. 1971.

[SS05] Yasushi Saito and Marc Shapiro. Optimistic replication. *Computing Surveys*, 37(1):42–81, March 2005.

[SS06] David Sun and Chengzheng Sun. Operation context and context-based operational transformation. In *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, CSCW '06, pages 279–288, New York, NY, USA, 2006. ACM.

[Sun02] Chengzheng Sun. Optional and responsive fine-grain locking in internet-based collaborative systems. *Parallel and Distributed Systems, IEEE Transactions on*, 13(9):994 – 1008, September 2002.

[SXS+06] Chengzheng Sun, Steven Xia, David Sun, David Chen, Haifeng Shen, and Wentong Cai. Transparent adaptation of single-user applications for multi-user real-time collaboration. *ACM Trans. Comput.-Hum. Interact.*, 13(4):531–582, December 2006.

[TH97] Dietmar Tutsch and Guenter Hommel. Performance of buffered multistage interconnection networks in case of packet multicasting. In *Proceedings of the 1997 Advances in Parallel and Distributed Computing Conference (APDC '97)*, APDC '97, pages 50–, Washington, DC, USA, 1997. IEEE Computer Society.

[TM03] Jonathan S. Turner and Riccardo Melen. Multirate clos networks. *IEEE Communications Magazine*, 41:38–44, 2003.

[Tom67] R. M Tomasulo. An efficient algorithm for exploiting multiple arithmetic units. *IBM Journal of Research and Development*, pages 25–233, 1967.

[Tur37] A. Turing. Computability and λ-definability. *Journal of Symbolic Logic*, 2:153–163, 1937.

[twi10] Twiki system requirements. available at: `http://twiki.org/cgi-bin/view/TWiki/TWikiSystemRequirements`, 2010.

[VP93] S. F. Vinokurov and N. A. Peryazev. A polynomial decomposition of boolean functions. *Mathematical Notes*, 53:130–133, 1993. 10.1007/BF01208315.

[WD98] R. Clint Whaley and Jack J. Dongarra. Automatically tuned linear algebra software. In *Supercomputing '98: Proceedings of the 1998 ACM/IEEE conference on Supercomputing (CDROM)*, pages 1–27, Washington, DC, USA, 1998. IEEE Computer Society.

[Wir96] Niklaus Wirth. *Compiler construction*. International computer science series. Addison-Wesley, 1996.

[WM91] M. E. Wolf and S. Lam. M. A data locality optimizing algorithm. In *The SIGPLAN'91 Conference on Programing Language Design and Implementation*, pages 30–44, June 1991.

[Wol89] M. J. Wolfe. More iteration space tiling. In *Supercomputing'89*, November 1989.

[WT00] Tilman Wolf and Jonathan S. Turner. Design issues for high performance active routers. In *Proc. of the International Zurich Seminar on Broadband Communications*, pages 199–205, Zurich, Switzerland, February 2000.

[WUM09] Stéphane Weiss, Pascal Urso, and Pascal Molli. Logoot: A scalable optimistic replication algorithm for collaborative editing on p2p networks. In *ICDCS*, pages 404–412. IEEE Computer Society, 2009.

[WUM10] Stéphane Weiss, Pascal Urso, and Pascal Molli. Logoot-undo: Distributed collaborative editing system on p2p networks. *IEEE Transactions on Parallel and Distributed Systems*, 21:1162–1174, 2010.

[YW02] Yuanyuan Yang and Jianchao Wang. A class of multistage conference switching networks for group communication. In *Proceedings of the 2002 International Conference on Parallel Processing*, ICPP '02, pages 73–, Washington, DC, USA, 2002. IEEE Computer Society.

[ZA02] B. Zhou and M. Atiquzzaman. A performance comparison of four buffering schemes for multistage interconnection networks, 2002.

Abstract

This thesis aims at developing strategies to advance the capabilities of sequential computations and distributed systems, particularly, it deals with sequential break down of operations and decentralized collaborative editing systems. In this thesis, we introduced precision control indexing method that generates unique identifiers which are used for indexed communication in distributed systems, particularly, in decentralized collaborative editing systems. These identifiers are still real numbers with a specific controlled pattern of precision. The set of identifiers is finite that facilitates in computing local as well as global cardinality. This property plays important role in dealing with indexed communication. In addition, we proved some other properties including order preservation. The indexing method is tested and verified through experimentation successfully and the method leads to design decentralized collaborative editing system. Dealing with sequential break down of operations, we explore limitations of the existing strategies, extended the idea by introducing new strategies. These strategies lead towards optimization (processor, compiler, memory, code). This style of decomposition attracts research communities for further investigation and practical implementation that could lead towards designing an arithmetic unit.

Keywords: collaborative editing, distributed computing, optimistic replication, algorithms, optimization (processor, memory), logic and computation, circuit design

Résumé

Cette thèse vise à développer des stratégies permettant d'augmenter la puissance du calcul séquentiel et des systèmes distribués, elle traite en particulier, la décomposition séquentielle des opérations ainsi que des systèmes d'édition collaboratifs décentralisés. Nous introduisons, une méthode d'indexage avec précision contrôlée. Celle-ci permet la génération d'identifiants uniques utilisés dans l'indexage des communications dans les systèmes distribués, plus particulièrement dans les systèmes d'édition collaboratifs décentralisés. Ces identifiants sont des nombres réels avec un motif de précision contrôlé. Un ensemble fini d'identifiants est conservé pour permettre le calcul de cardinalités locales et globales. Cette propriété joue un rôle prépondérant dans la gestion des communications indexées. De plus, d'autres propriétés incluant la préservation de l'ordre sont observées. La méthode d'indexage a été testée et vérifiée avec succès. Ceci a permis la conception d'un système d'édition collaboratif décentralisé. Aussi, nous explorons les stratégies existantes, relatives

a la décomposition séquentielle d'opérations, que nous étendons à de nouvelles stratégies. Ces stratégies mènent à une optimisation (processeur, compilateur, mémoire, code). Ces styles de décomposition portent un intérêt majeur à la communauté scientifique. Des recherches et des implémentations de plus en plus rapides résultent de la conception d'unité arithmétique.

Mots-clés: édition collaborative, modèle de cohérence des données, structure de données persistante, P2P, CRDT, réplication optimiste, algorithmes, optimisation de la mémoire, l'optimisation de processeur, la logique et de calcul, la conception de circuits