이학 박사 학위논문

Analyses of the Perfect Distinguished Point Tradeoff and Its Parallel Treatment

(중복제거 테이블을 이용한 특이점 절충기법과 그의 병렬처리에 대한 분석)

2016년 2월

서울대학교 대학원

수리과학부

이가원

Analyses of the Perfect Distinguished Point Tradeoff and Its Parallel Treatment

(중복제거 테이블을 이용한 특이점 절충기법과 그의 병렬처리에 대한 분석)

지도교수 홍 진

이 논문을 이학 박사 학위논문으로 제출함

2015년 10월

서울대학교 대학원

수리과학부

이가원

이가원의 이학 박사 학위논문을 인준함

2015년 12월

위 원 장	______________________	(인)
부 위 원 장	______________________	(인)
위 원	______________________	(인)
위 원	______________________	(인)
위 원	______________________	(인)

Analyses of the Perfect Distinguished Point Tradeoff and Its Parallel Treatment

A dissertation
submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
to the faculty of the Graduate School of
Seoul National University

by

Ga Won Lee

Dissertation Director : Professor Jin Hong

Department of Mathematical Sciences
Seoul National University

February 2016

Abstract

Analyses of the Perfect Distinguished Point Tradeoff and Its Parallel Treatment

Ga Won Lee
Department of Mathematical Sciences
The Graduate School
Seoul National University

In a recent paper, the performances of three major time memory tradeoff algorithms, namely, the classical Hellman tradeoff and the non-perfect table versions of the distinguished point(DP) and the rainbow table tradeoff methods, were analyzed and compared against each other. The analysis was accurate in the sense that the extra costs of resolving false alarms were not ignored, and the performance comparison was fair in the sense that both the online complexity and the pre-computation cost were taken into account and the techniques for optimizing storage size were taken into account. Based on this paper, another recent paper analyzed a DP variant, which treats the non-perfect DP tables in parallel, and compared its performance with those of the previous three tradeoff algorithms. In this thesis, we analyze the performances of three more tradeoff algorithms and compare them with the aforementioned four algorithms. The algorithms newly considered here will be the perfect table versions of the DP, rainbow table, and parallel DP tradeoff methods.

The performance of an algorithm cannot be represented by a single numeric value and algorithm preferences will depend on the available resources and various situations faced by the tradeoff algorithm implementer. Hence, we will present the performances of the tradeoff algorithms as curves providing the full range of options made available by the algorithms, so as to allow for the implementers to make their choices. However, our comparisons show that, under typical situations, the perfect table parallel DP tradeoff algorithm is more likely to be preferable over the other DP algorithm variants and that the perfect rainbow table method is superior to the other tradeoff algorithms.

On the other hand, yet another recent paper notes that the perfect rainbow table method is widely implemented in practice to process its pre-computation tables in a serial manner, rather than in parallel, as was originally proposed by the algorithm designers. This is because, even though the parallel treatment of the pre-computation tables would be more efficient in theory, the size of tables are too large to be fully loaded into fast main memory in real-world applications such as password recovery and this affects the real-world performances of the algorithms negatively. Following the approach of the paper, we give the optimal physical wall-clock online execution times for the practically used serial perfect rainbow and the perfect table versions of the DP and rainbow tradeoffs that treat their pre-computation tables in parallel. This is done with various realistic password spaces and at various high success rate requirements, under a specific limitation on the size of available storage. Unlike any theoretical approach to the tradeoff algorithms, the physical online execution time includes the time taken for loading the pre-computation tables from disk to fast memory and the time taken by table lookups.

We find that, in contrast with the software developers' intuition, the serial perfect rainbow tradeoff algorithm is inferior to the two algorithms that treat their tables in parallel, when their optimal physical online times are compared under reasonable assumptions and settings. Our simplified conclusions are that, for the larger of the two search spaces we dealt with, the parallel version of the perfect rainbow table method gives the shortest wall-clock online time, and that, for the smaller search space, when restricted to the same amount of pre-computation, the perfect parallel DP tradeoff is faster than the other algorithms.

Key words: time memory tradeoff, distinguished point, rainbow table, perfect table, algorithm complexity

Student Number: 2011-30898

Contents

Chapter 1

Introduction

Cryptanalytic time memory tradeoff is a tool for quickly inverting a one-way function, utilizing some pre-computed data specific to the one-way function. It is widely used to recover passwords from known password hashes. A tradeoff algorithm consists of two parts, the pre-computation and online phases. In the pre-computation phase, which is performed before any inversion target, such as a password hash, is given, a massive amount of one-way function iterations is performed and a digest of the pre-computed data is stored, which is referred to as the pre-computation tables. When an inversion target is given, the algorithm performs further computations related to the target, in order to return the correct inverse of the given target with a pre-fixed probability of success, utilizing the pre-computed data, and this process is referred to as the online phase.

Time memory tradeoff was first introduced by Hellman [13] and many variations of Hellman's original tradeoff algorithm are available today. The most widely known and used tradeoff algorithms are the distinguished point variant [10, 11] and the rainbow table method [24]. We will refer to these algorithms as the DP tradeoff and the rainbow tradeoff, respectively. Both of these algorithms have two subversions that treat the non-perfect tables and perfect tables. Each of them can also be further split into two subversions that process their (non-perfect or perfect) tables in serial and in parallel. Parallel processing of DP tables was mentioned in [10, 11, 26] and it is widely known that processing small numbers of rainbow tables in parallel can reduce the total combined processor time [24]. Analysis of the DP tradeoff which processes non-perfect DP tables in parallel was presented in the previous work [16], which concluded that parallel processing of the non-perfect DP tables is better than its serial processing, but that both fall behind the non-perfect rainbow tradeoff.

An ultimate common goal for analyses of tradeoff algorithms is to compare the algorithm performances against each other so as to determine which tradeoff is superior to the others. In doing this, choosing a reasonable and fair method of comparison is very important. We will take two different approaches in comparing the tradeoff algorithms, and brief explanations of these are given below.

The first method of tradeoff comparison that we take was recently suggested by [18]. Existing analyses [6, 8, 12, 13] show that most tradeoff algorithms satisfy a tradeoff curve of the form

$$TM^2 \approx \mathsf{N}^2,$$

where $\frac{TM^2}{\mathsf{N}^2}$ is some numeric value, referred to as the *time memory tradeoff coefficient*, that is neither very large nor close to zero. Here, T is the online execution time, M is the storage size required to store the pre-computation tables, and N is the size of the space that the one-way function acts on. The work [18] calculated the accurate formulas for the tradeoff coefficients of the classical Hellman, non-perfect (serial) DP, and non-perfect (parallel) rainbow tradeoffs and found that all of them may be expressed as functions of two variables, the success rate and a certain algorithm parameter. The online behavior of each tradeoff algorithm can be presented as the tradeoff curve, represented by (T, M)-pair options, and the options are made by the time memory tradeoff coefficient which can be computed from a fixed success rate requirement and algorithm parameters. In other words, one can make a trade-off between the online execution time T and the required online memory M from the time memory tradeoff coefficient, determined by the success rate and algorithm parameter. Thus, the tradeoff coefficient $\frac{TM^2}{\mathsf{N}^2}$ can be accepted as a measure of how efficient a tradeoff algorithm is, with a smaller coefficient indicating a more efficient algorithm.

However, for a fixed success rate, algorithm parameters that lead to a smaller tradeoff coefficient usually calls for a larger pre-computation effort. Hence, one must work with further tradeoffs between the online efficiency and pre-computation cost, at any fixed success requirement. Thus, it is hard to clarify what it means to select the parameters achieving the *optimal* performance of a tradeoff algorithm.

The work [18] let the algorithm implementers make the final judgment in choosing a tradeoff algorithm and the appropriate balance between pre-computation cost and online efficiency based on their requirements, such as success rate, online efficiency, and available pre-computation and online resources. At a fixed success rate, the secondary tradeoffs between the tradeoff coefficient and pre-computation cost can be presented as a curve for each algorithm. Thus, at a common success rate, algorithm comparison can be made by plotting those curves on a plane, after certain precautions are made to account for differences in the number of bits required to record each table entry by the algorithms. Then, the tradeoff implementer can choose an algorithm together with the online efficiency and pre-computation cost pair, provided by that algorithm, based on his or her circumstances and requirements.

Analyses presented in [16, 18, 21] did not deal with the perfect serial DP, perfect parallel DP, and perfect parallel rainbow tradeoffs. Here, we compute the time memory tradeoff coefficients together with other necessary information to plot the graphs for these three tradeoffs. Our simplified conclusion is that at most success rates, the perfect parallel DP tradeoff outperforms the other three DP tradeoffs, namely, the non-perfect serial DP, non-perfect parallel DP, and perfect serial DP tradeoffs, in typical situations. Similarly, when the perfect parallel DP tradeoff is compared with the non-perfect and perfect parallel rainbow tradeoffs, we can essentially conclude that, in typical cases, the perfect parallel rainbow tradeoff is likely to be preferable over the other two tradeoffs. However, as mentioned above, the favored tradeoff algorithm may be different for each implementer's situation.

Let us next discuss our second approach to algorithm performance comparison, the one that is based on the physical wall-clock online execution times of the tradeoff algorithms. The previous analyses for the parallel versions of tradeoff algorithms [16, 18] had ignored the costs of lookups to the pre-computation tables, under assumption that all of the pre-computation tables reside in fast main memory. However, to find the pre-image of a password hash created from a reasonably long real-world password, the search space size N is required to be so large that loading all the pre-computation tables into fast main memory and treating them in parallel is typically impossible. For this reason, as was noted by the recent work [21], even though the parallel processing of rainbow tables is widely considered to be more efficient than its serial processing in theory [24], real-world implementations of the rainbow tradeoff [1,3] treat the pre-computation tables serially.

In our practical comparison of the parallel versions of the perfect DP and perfect rainbow tradeoffs with the serial version of the perfect rainbow tradeoff, we assume that all tradeoffs execute their online phases with the pre-computation tables initially located on the slow long-term storage. Also, the effort of loading table entries into fast memory taken by the serial perfect rainbow tradeoff and the time taken by the parallel perfect DP and rainbow tradeoffs in accessing the tables are no longer ignored. Taking these factors into account, we present the explicit physical online execution times for the above mentioned three tradeoff algorithms, combining constants concerning the speeds of the one-way function computations, loading of tables, and table lookups that are set to realistic figures through our test measurements.

We could conclude from the formulas that, for the two parallel tradeoffs under consideration at a fixed success rate, each storage size M determines a minimum physical online time and that this minimum time becomes smaller as the storage size M is increased. On the other hand, for the serial perfect rainbow tradeoff, there exists an M value that provides the minimum physical online time, so that, once the available storage size reaches this M value, the physical online time cannot be improved further. We will refer to the shortest physical online time for each tradeoff algorithm as the *optimal* physical online time.

When realistic search spaces and a 4 TB hard disk drive for pre-computation table storage are assumed, at high success rates, such as 99.9%, 99%, and 90%, the optimal online time of the serial perfect rainbow tradeoff is much larger than those of the two parallel tradeoffs under consideration. Comparing the perfect parallel DP and the perfect parallel rainbow tradeoffs against each other, we could see that, for the larger of the two search spaces we dealt with, the perfect rainbow tradeoff has a smaller optimal physical online time and even achieves this with a lower pre-computation cost. However, when a smaller, but still realistic, search space is assumed, the perfect parallel DP tradeoff requires shorter time in performing the online phase than the perfect rainbow tradeoff, under the same pre-computation investment.

The rest of this thesis is organized as follows. In the next chapter, we fix the terminology, clarify the exact versions of the algorithms and the storage optimization techniques we will be dealing with, and review existing analyses of the non-perfect DP and non-perfect rainbow tradeoffs. In Chapter 3 and Chapter 4, the execution behaviors of the perfect serial DP, perfect parallel rainbow, and perfect parallel DP tradeoffs are analyzed. The analyses include the computations of the expected online time complexities that do not ignore the effects of false alarms and the time memory tradeoff coefficients. Storage optimization techniques are also discussed for each tradeoff algorithm and the correctness of our theoretical developments is also verified through experiments. The analysis for the perfect parallel rainbow tradeoff is totally based on previous results and contains no new ideas. The results of Chapter 3 were previously published through [22]. In Chapter 5, we review the method of tradeoff comparison that was suggested in [18] and compare the four DP variants against each other. After this, the most powerful

DP tradeoff, the perfect parallel DP tradeoff, is compared against the non-perfect parallel and perfect parallel rainbow tradeoffs. Chapter 6 is devoted to computing the optimal physical online times for the parallel perfect DP, parallel perfect rainbow, and the serial perfect rainbow tradeoffs. To do this, we compute the expected number of table lookups for the perfect parallel DP and perfect rainbow tradeoffs, review existing analysis of the serial perfect rainbow tradeoff, and then express and find the minimum of the physical online time for each tradeoff algorithm. Combining realistic constants, the optimal physical online times for the tradeoffs are computed and compared against each other, at various situations. Finally, all the results are summarized in Chapter 7. Appendix contains descriptions and results of our measurements done to decide the realistic constants that were used in our practical comparisons.

Chapter 2

Preliminaries

2.1 Algorithm Clarification, Terminology, and Notation

This section aims to make this paper self-contained, but the reader may still find it helpful to refer to [18] for more detail. The work [18] also clarifies many obscure technical details that are not discussed elsewhere in the related literature, which should be of interest to the mathematically oriented cryptographers.

Throughout this paper, the function $F : \mathcal{N} \to \mathcal{N}$ will always act on a search space $\mathcal{N}$ of size N. In practical applications, the function F is the specific one-way function to be inverted, which has a co-domain that is much larger than the domain. However, any theoretical analysis of the tradeoff algorithms will treated it as a random function with matching domain and co-domain, and we will do the same throughout this work. The symbol $\mathbf{p}$ will be used in this thesis to denote the unknown answer that is to be recovered by a tradeoff algorithm, and the inversion target will be denoted by $\mathbf{h} = F(\mathbf{p})$. The reader could think of these two as the *password* to be recovered and the password *hash* given as the inversion target. As is customary with cryptology papers, the symbol $\log n$ will denote $\log_2 n$.

2.1.1 Four Versions of the DP Tradeoff

To setup any variants of the DP tradeoff, one first fixes positive integer parameters m, t, and ℓ. The integers must be chosen to satisfy the *matrix stopping rule* $mt^2 \approx \mathsf{N}$ and the relation $\ell \approx t$. Theoretical analyses of the DP tradeoff often focus on the choice of $m \approx t \approx \ell \approx \mathsf{N}^{\frac{1}{3}}$, because this choice minimizes the overall complexity, defined as the sum of the online time and storage complexities. This set of parameters, with a *lot* of flexibility allowed for the approximate conditions, is being assumed when we refer to the DP tradeoff executed in a *typical environment*.

Once the parameters are fixed, one chooses a certain characteristic that is satisfied by a random element of $\mathcal{N}$ with probability $\frac{1}{t}$. This *distinguishing property* must be extremely easy to check for elements of $\mathcal{N}$. Any element of $\mathcal{N}$ satisfying the distinguishing property is called a *distinguished point* (DP). A typical approach is to set t to a power of 2 and to define all points of $\mathcal{N}$ with $\log t$ leading zeros as DPs. Next, bijections $rd_i : \mathcal{N} \to \mathcal{N}$ $(i = 1, \ldots, \ell)$ are fixed to define the i-th *colored* one-way functions $F_i = rd_i \circ F$. The *reduction functions* rd_i are chosen so that they require negligible resources to compute. A typical approach is to have each reduction function XOR a fixed constant to its input, with the constants chosen to be distinct for each i. Finally, another positive integer $\vec{m}_0$, that is correlated to m (and t) in a manner to be explained below for each tradeoff algorithm, is fixed.

Original DP tradeoff `D` We shall refer to the DP variant of the Hellman's original algorithm [10, 11] as the *original DP tradeoff* `D`.

The pre-computation and online phase of the original DP tradeoff are given by Algorithm 1 and Algorithm 2, respectively. Further details will be explained below, together with the terminology and notation to be used in this work.

The integer $\vec{m}_0$ of Algorithm 1 must be set so that each table $\mathtt{DT}_i$ is expected to contain m entries. According to [18], $\vec{m}_0 \approx m$ is appropriate when a sufficiently large chain length upper bound is used.

Algorithm 1: Pre-computation phase of D and pD

```
for $i = 1, \ldots, \ell$ do
    DT$_i \leftarrow \emptyset$;                                  // DP table
    for $j = 1, \ldots, \vec{m}_0$ do
        Choose sp$^i_j \in \mathcal{N}$;                      // starting point
        tp $\leftarrow F_i($sp$^i_j)$;
        while tp is not a DP do   // generate pre-computation chain
            tp $\leftarrow F_i($tp$)$;
        end
        ep$^i_j \leftarrow$ tp;                                // ending point
        Append $($sp$^i_j$, ep$^i_j)$ to DT$_i$;
    end
    Sort DT$_i$ according to the ep's;
    Record result to disk as DT$_i$;
end
```

Algorithm 2: Online phase of D

```
for $i = 1, \ldots, \ell$ do
    op $\leftarrow rd_i(\mathbf{h}) = F_i(\mathbf{p})$;
    while op is not a DP do                    // generate online chain
        op $\leftarrow F_i($op$)$;
    end
    Search for op among the ep's of DT$_i$;
    if Matches are found then
        for each pair of $($sp$^i_j$, ep$^i_j)$'s in DT$_i$ with a common ep$^i_j$==op do
            tp$_{\text{new}} \leftarrow F_i($sp$^i_j)$;
            while [tp$_{\text{new}}$ != $rd_i(\mathbf{h})$] and [tp$_{new}$ is not a DP] do
            // regenerate pre-comp chain
                tp$_{\text{old}} \leftarrow$ tp$_{new}$;
                tp$_{\text{new}} \leftarrow F_i($tp$_{\text{old}})$;
            end
            if $F($tp$_{\text{old}})$ == $\mathbf{h}$ then                // answer found
                return tp$_{\text{old}}$ as answer and terminate;
            end
        end
    end
end
```

The table $\mathtt{DT}_i = \{(\mathtt{sp}^i_j, \mathtt{ep}^i_j)\}_{j=1}^{\vec{m}_0}$ produced by Algorithm 1 is called the i-th *non-perfect DP table*. For a non-perfect DP table, any collection of starting point and ending point pairs could share common ending point. Namely, a non-perfect DP table is not necessarily free of duplicates among its ending points.

A series of elements from $\mathcal{N}$, obtained through iterated applications of F, or some F_i, that ends at its first occurrence of a DP is a *DP chain*. Since a DP occurs with probability $\frac{1}{t}$, the expected length of a randomly generated DP chain is t. Each series of points spanning from a $\mathtt{sp}^i_j$ to the corresponding $\mathtt{ep}^i_j$ is a *pre-computation DP chain*, and the first part of Algorithm 2 generates an *online DP chain*. We take the convention that an online chain starts from the unknown answer $\mathbf{p}$, rather than from the inversion target $\mathbf{h}$ or $rd_i(\mathbf{h})$.

The collection of all pre-computation DP chains corresponding to a pre-computation DP table is a *DP matrix*. The use of the terms table and matrix in this thesis are not interchangeable, and the reader should exercise care to distinguishing the two. The matrices corresponding to $\mathtt{DT}_i$ will be referred to as the *non-perfect DP matrix* $\mathtt{DM}_i$.

As the online chain corresponding to a pre-computation table is generated, the chain will either reach a DP that does not reside in $\mathtt{DT}_i$ or merge into (greater than or equal to) one of the pre-computation chains of $\mathtt{DM}_i$. Because the function F_i that is being iterated is not injective, the discovery of $\mathtt{op} == \mathtt{ep}^i_j$ does not guarantee that the answer $\mathbf{p}$ to the inversion problem will be recovered through the regeneration of the corresponding pre-computation chain that starts from $\mathtt{sp}^i_j$, and these instances are called *false alarms*. Any ending point match $\mathtt{op} == \mathtt{ep}^i_j$ is an *alarm*, and the bottom half of Algorithm 2 works to *resolve* this alarm.

To summarize, the non-perfect DP tradeoff which treat its pre-computation tables in a serial manner will be referred to as the original DP tradeoff $\mathtt{D}$, and the *non-perfect DP tradeoff* will be also used to represent the original DP tradeoff.

Parallel DP tradeoff `pD` We shall refer to the original DP tradeoff that processes non-perfect DP tables in parallel as the *parallel DP tradeoff* `pD`, analyzed in the previous work [16], and we will use the formulas of [16] when comparing performances of tradeoff algorithms. The specific version of the `pD` tradeoff, treated in [16] is described below.

The pre-computation and online phases of the `pD` tradeoff are given by Algorithm 1 and Algorithm 3, respectively. As for the original DP tradeoff, $\vec{m}_0 \approx m$ when a sufficiently large chain length upper bound is used , because the parallel DP tradeoff shares the pre-computation phase with the original DP tradeoff. Same terminology and notation related to the pre-computation phase will be used, such as a non-perfect DP table `DT`, a non-perfect DP matrix `DM`, and pre-computation DP chain. Also, since online phase is slightly different with the original DP tradeoff when treating pre-computation tables, the terminology, such as online chain, alarm, and false alarm, is maintained. Further details and certain tweaks to the algorithms that should be assumed will be explained below.

The work [14, 26] suggested that all the pre-computation tables be processed in parallel, rather than sequentially, during the online phase. Parallel processing causes the shorter online chains to be treated before the longer ones, and since the online phase is likely to terminate with the correct answer before any of the pre-computation tables are fully processed, this leads to a larger portion of the online computation being spent on processing the shorter chains. Since shorter chains are less likely to induce false alarms, this has a positive effect of reducing the cost of dealing with alarms. Predicting its positive effect had shown in the previous work [16].

Let us explain details of Algorithm 3. The number of DP tables is roughly of $\mathsf{N}^{\frac{1}{3}}$ order, which is likely to be larger than the number of available processors, implying that each processor will be assigned to multiple tables. In such a situation, we require each processor to work with its share of assigned tables in a round-robin fashion. A processor should process a single iteration for a table and then

Algorithm 3: Online phase of pD

```
for $i = 1, \ldots, \ell$ do                    // Initialize every online chain
    $\texttt{op}_i \leftarrow rd_i(\mathbf{h}) = F_i(\mathbf{p})$;
end
repeat
    for $i = 1, \ldots, \ell$ do
        if $\texttt{op}_i$ is a DP then
            Search for $\texttt{op}_i$ among the ep's of $\bar{\texttt{DT}}_i$;
            if Matches are found then
                for each pair of $(\texttt{sp}^i_j, \texttt{ep}^i_j)$'s in $\texttt{DT}_i$ with a common $\texttt{ep}^i_j$==$\texttt{op}_i$
                do
                    $\texttt{tp}_{\text{new}} \leftarrow F_i(\texttt{sp}^i_j)$;
                    while [$\texttt{tp}_{\text{new}}$ != $rd_i(\mathbf{h})$] and [$\texttt{tp}_{new}$ is not a DP] do
                    // regenerate pre-comp chain
                        $\texttt{tp}_{\text{old}} \leftarrow \texttt{tp}_{new}$;
                        $\texttt{tp}_{\text{new}} \leftarrow F_i(\texttt{tp}_{\text{old}})$;
                    end
                    if $F(\texttt{tp}_{\text{old}})$ == h then          // answer found
                        return $\texttt{tp}_{\text{old}}$ as answer and terminate;
                    end
                end
            end
        end
        else
            $\texttt{op}_i \leftarrow F_i(\texttt{op}_i)$;
        end
    end
until All $\texttt{op}_i$'s meet DPs;
```

move onto the next table it was assigned, rather than take the approach of fully processing one table and then fully processing its next assigned table.

We have partially clarified how DP should be parallelized, but there still is an issue concerning the resolving of false alarms. Consider, for the moment, a fully parallel system, where all the DP tables are distributed to different processors. When a processor encounters an alarm, it will regenerate a pre-computation chain, during which time period other processors will continue with their respective online chain iterations. By the time the alarm is resolved, many of the other processors would have reached the end of the online chain creation. This shows that the approach trying to have more time spent on short online chains fails in the fully parallel environment.

Fortunately, each processor is likely to be assigned multiple tables in practice. We assume that each processor is made to resolve any alarm that it encounters, before processing any more online chain iterations. Then, since each processor will be struggling to resolve its share of alarms, further iterations of the online chains are effectively postponed until many of the alarms are resolved. If a set of tables allocated to a certain processor rarely produces alarms, online chain iterations for this set of tables will proceed faster than those of other sets of tables, but the overall behavior will be as if the online chain iterations were delayed until current alarms are resolved.

During analysis of algorithm treating pre-computation tables in parallel, when counting the total function iterations, we shall take the simplified view that the i-th online chain iterations for all tables are executed simultaneously and that the $(i+1)$-th simultaneous iterations are executed only after all alarms encountered at the i-th iterations are resolved. This view correctly reflects the parallelization details discussed so far.

We also assume that Algorithm 3 is slightly modified so as to incorporate the *online chain record technique* [14, 26]. While generating an online chain, one keeps track of not just the current foremost point $\mathtt{tp}_{\text{new}}$ of the chain, but keeps a record of all the generated intermediate points. When resolving an alarm, one compares the current end of the regenerated pre-computation chain against the complete online chain, rather than just the reduced inversion target point $rd_i(\mathbf{h})$, so that one may stop the pre-computation chain regeneration at the exact position of chain merge, rather than at the ending point DP. Thus, online chain record will surely increase the efficiency of the tradeoff algorithm.

Perfect DP tradeoff $\bar{\mathtt{D}}$ We shall refer to the DP tradeoff which treats perfect DP tables in a serial manner as the *perfect DP tradeoff* $\bar{\mathtt{D}}$.

The pre-computation and online phases of the perfect DP tradeoff, in their rudimentary forms, are given by Algorithm 4 and Algorithm 5, respectively. Further details and certain tweaks to the algorithms that should be assumed will be explained below, together with the terminology and notation to be used in this paper.

Algorithm 4: Pre-computation phase of $\bar{\mathtt{D}}$ and $\bar{\mathtt{pD}}$

```
for $i = 1, \dots, \ell$ do
    $\mathtt{TDT}_i \leftarrow \emptyset$;                                  // temporary DP table
    for $j = 1, \dots, \vec{m}_0$ do
        Choose $\mathtt{sp}^i_j \in \mathcal{N}$;                           // starting point
        $\mathtt{tp} \leftarrow F_i(\mathtt{sp}^i_j)$; $\mathsf{len}_j \leftarrow 1$;
        while tp is not a DP do  // generate pre-computation chain
            $\mathtt{tp} \leftarrow F_i(\mathtt{tp})$; $\mathsf{len}_j \leftarrow \mathsf{len}_j + 1$;
        end
        $\mathtt{ep}^i_j \leftarrow \mathtt{tp}$;                                  // ending point
        Append $(\mathtt{sp}^i_j, \mathtt{ep}^i_j, \mathsf{len}_j)$ to $\mathtt{TDT}_i$;
    end
    Sort $\mathtt{TDT}_i$ according to the ep's;
    for each group of $(\mathtt{sp}, \mathtt{ep}, \mathsf{len})$'s in $\mathtt{TDT}_i$ with a common ep do
    // remove ep collisions
        Discard all triples except for the one with the largest len;
    end
    Remove all len information from $\mathtt{TDT}_i$ and record result to disk as $\bar{\mathtt{DT}}_i$;
end
```

Algorithm 5: Online phase of $\bar{\mathtt{D}}$

```
for $i = 1, \dots, \ell$ do
    $\mathtt{op} \leftarrow rd_i(\mathbf{h}) = F_i(\mathbf{p})$;
    while $\mathtt{op}$ is not a DP do                    // generate online chain
        $\mathtt{op} \leftarrow F_i(\mathtt{op})$;
    end
    Search for $\mathtt{op}$ among the $\mathtt{ep}$'s of $\bar{\mathtt{DT}}_i$;
    if $\mathtt{op} == \mathtt{ep}^i_j$ then
        $\mathtt{tp}_{\mathrm{new}} \leftarrow F_i(\mathtt{sp}^i_j)$;
        while $[\mathtt{tp}_{\mathrm{new}} \mathrel{!=} rd_i(\mathbf{h})]$ and $[\mathtt{tp}_{new}$ is not a DP$]$ do
        // regenerate pre-comp chain
            $\mathtt{tp}_{\mathrm{old}} \leftarrow \mathtt{tp}_{new}$;
            $\mathtt{tp}_{\mathrm{new}} \leftarrow F_i(\mathtt{tp}_{\mathrm{old}})$;
        end
        if $F(\mathtt{tp}_{\mathrm{old}}) == \mathbf{h}$ then                  // answer found
            return $\mathtt{tp}_{\mathrm{old}}$ as answer and terminate;
        end
    end
end
```

For the perfect DP tradeoff, the integer $\vec{m}_0$ of Algorithm 4 must be set so that each table $\bar{\mathtt{DT}}_i$ is expected to contain m entries having no common ending points. The appropriate value for $\vec{m}_0$ is given later in the next chapter by Lemma 3.2, as a function of m and t. An alternative method is to modify the algorithm to incrementally add more starting points until the number of distinct ending points reaches m, but we will not treat this approach.

The table $\bar{\mathtt{DT}}_i$ produced by Algorithm 4 is called the i-th *perfect DP table* and the larger auxiliary table $\mathtt{DT}_i = \{(\mathtt{sp}^i_j, \mathtt{ep}^i_j)\}_{j=1}^{\vec{m}_0}$ is referred to as the *non-perfect DP table*, *corresponding to the perfect table* $\bar{\mathtt{DT}}_i$. Note that, even though *any* collection of starting point and ending point pairs that contains no duplicates among its ending points could be called a *perfect* table and any similar collection that is not necessarily free of duplicates could be called a *non-perfect* table, our reference to perfect and non-perfect DP tables in this paper will almost always be to the tables $\bar{\mathtt{DT}}_i$ and $\mathtt{DT}_i$ that result from an execution of Algorithm 4. We will not be discussing properties of perfect DP tables created in any other manner.

Analogous to the previous two DP tradeoffs, the matrix corresponding to $\bar{\mathtt{DT}}_i$ be referred to as the *perfect DP matrix* $\bar{\mathtt{DM}}_i$. Likewise, the collection of approximately $\vec{m}_0$ chains corresponding to $\mathtt{DT}_i$ is the *non-perfect DP matrix* $\mathtt{DM}_i$, *corresponding to the perfect DP matrix* $\bar{\mathtt{DM}}_i$.

It is helpful to visualize a DP matrix as a directed graph with arrows representing the actions of the one-way function. With this view, one may state that a non-perfect DP matrix contains many chains that *merge* into each other and that a perfect DP matrix is free of chain merges. The chains are usually visualized as having been laid out in the horizontal direction with arrows point from left to right, so that a perfect DP matrix contains *m rows* of various lengths that do not merge into each other with the starting points on the left and the ending points on the right.

The collision removal process of Algorithm 4 discards all chains except for the longest one from each group of merging chains found in a non-perfect DP matrix. Retaining the longest chain [10, 11] is the standard approach, as this is expected to be beneficial to the success rate of the online phase. A merge of chains of equal length is a rare event that need not be considered during our analysis.

It may safely be said that the original and perfect DP tradeoff algorithms share the common online phase. When alarm occurs, perfect table guarantees that a single pre-computation chain regeneration is necessary to verify the alarm, as can be seen in Algorithm 5, because $\bar{\mathtt{DT}}$ contains the only one corresponding entry that has same ending point with the online DP chain. From this point of view, one can expect that use of perfect tables bring positive effect on online performance of the tradeoff algorithm.

In addition, as for the parallel DP tradeoff, the online chain recording technique to reduce the effort of resolving alarms is utilized when analyzing the online complexity of the perfect DP tradeoff, which will be presented in Section 3.1

Perfect Parallel DP tradeoff $\bar{\mathtt{pD}}$ The parallel version of the perfect DP tradeoff will be referred to as the *perfect parallel DP tradeoff* $\bar{\mathtt{pD}}$. The pre-computation and online phases of the perfect parallel tradeoff are given by Algorithm 4 and Algorithm 6, respectively. As previous two DP tradeoffs, the online chain recording technique is applied to the online phase for reducing a number of chain walk steps when resolving alarms.

Algorithm 6: Online phase of $\bar{\mathtt{pD}}$

```
for $i = 1, \dots, \ell$ do                          // Initialize every online chain
    $\mathtt{op}_i \leftarrow rd_i(\mathbf{h}) = F_i(\mathbf{p})$;
end
repeat
    for $i = 1, \dots, \ell$ do
        if $\mathtt{op}_i$ is a DP then
            Search for $\mathtt{op}_i$ among the ep's of $\bar{\mathtt{DT}}_i$;
            if $\mathtt{op}_i == \mathtt{ep}^i_j$ then
                $\mathtt{tp}_{\text{new}} \leftarrow F_i(\mathtt{sp}^i_j)$;
                while [$\mathtt{tp}_{\text{new}}$ != $rd_i(\mathbf{h})$] and [$\mathtt{tp}_{new}$ is not a DP] do
                // regenerate pre-comp chain
                    $\mathtt{tp}_{\text{old}} \leftarrow \mathtt{tp}_{new}$;
                    $\mathtt{tp}_{\text{new}} \leftarrow F_i(\mathtt{tp}_{\text{old}})$;
                end
                if $F(\mathtt{tp}_{\text{old}}) == \mathbf{h}$ then         // answer found
                    return $\mathtt{tp}_{\text{old}}$ as answer and terminate;
                end
            end
        end
        else
            $\mathtt{op}_i \leftarrow F_i(\mathtt{op}_i)$;
        end
    end
until All $\mathtt{op}_i$'s meet DPs;
```

As with the perfect DP tradeoff, the parallel DP tradeoff and the $\bar{\mathtt{pD}}$ tradeoff share the identical online phase except a number of the pre-computation regeneration to resolve a single alarm.

Note that a recent talk [26] announced the $\mathtt{p\bar{D}}$ tradeoff as the "New World Champion" of tradeoff algorithms, based on experimental results. Thus, it is meaningful that the performance of the $\mathtt{p\bar{D}}$ tradeoff is analyzed and compared against the well-known most efficient tradeoff algorithm, the rainbow table method, to verify the claim.

The terminology and notation of the perfect DP tradeoff, such as the perfect DP table $\mathtt{D\bar{T}}$, its corresponding non-pefect DP table $\mathtt{DT}$, perfect DP matrix $\mathtt{DM}$, and its corresponding non-perfect DP matrix $\mathtt{D\bar{M}}$ will be used identically, because the $\mathtt{p\bar{D}}$ tradeoff treats perfect DP tables produced by the identical pre-computation phase with the perfect DP tradeoff.

Any implementation of the DP tradeoff will introduce an upper bound $\hat{t}$ on the lengths of pre-computation and online chains to deal with chains falling into loops [10, 11]. That is, the while-loop of Algorithm 1 and Algorithm 4, the first while-loop of Algorithm 2 and Algorithm 5, and repeat-loop of Algorithm 3 and Algorithm 6 need to be augmented with another condition to prevent occurrences of infinite loops. A lower bound $\check{t}$ can also be used [25, 27] to discard short pre-computation chains that contribute little to the search space coverage. In this paper, no lower bound and a sufficiently large upper bound, such as $\hat{t} = 15t$, on the chain lengths are assumed. This simplifies our theoretical developments by ensuring that the possibility of an online chain not meeting the chain length bound conditions will be negligible, and also by allowing us to ignore the effects of discarding long or short pre-computation chains.

In the rest of this paper, we will mostly be focusing on a single DP matrix or table. This is only natural, as no argument can be specific to a reduction function rd_i. Hence, the table index i will be dropped from all notation and the simplified symbols $\mathtt{D\bar{T}}$, $\mathtt{DT}$, $\mathtt{D\bar{M}}$, and $\mathtt{DM}$ will be used. Likewise, the iteration function F_i will be written simply as F. The k-times iterated composition $F \circ \cdots \circ F$ of function F (or F_i) will be written as F^k.

We will use the notation $\mathtt{X}_{\mathrm{msc}} = \frac{mt^2}{\mathsf{N}}$ to represent the matrix stopping rule for each DP tradeoff algorithm, referred to as the *matrix stopping constant*. It should be careful that when dealing with the perfect table variant equivalent of $\bar{\mathtt{D}}_{\mathrm{msc}} = \frac{mt^2}{\mathsf{N}}$ or $\mathtt{p}\bar{\mathtt{D}}_{\mathrm{msc}} = \frac{mt^2}{N}$, the corresponding non-perfect variant equivalent of $\mathtt{D}_{\mathrm{msc}} = \frac{\vec{m}_0 t^2}{\mathsf{N}}$ will be used, because a perfect matrix contains the information about the associated non-perfect matrix having $\vec{m}_0$ entries before chain removal process of Algorithm 4.

The *coverage rates* $\mathtt{D}_{\mathrm{cr}}$ and $\bar{\mathtt{D}}_{\mathrm{cr}}$ of a non-perfect DP matrix $\mathtt{DM}$ and a perfect DP matrix $\bar{\mathtt{DM}}$, are defined to be the expected number of distinct nodes $|\mathtt{DM}|$ and $|\bar{\mathtt{DM}}|$ in the non-perfect DP matrix and the perfect DP matrix, divided by mt, respectively. More precisely, only the points that are used as inputs to the one-way function are counted, so that the ending point DPs are excluded in the count $|\mathtt{DM}|$ and $|\bar{\mathtt{DM}}|$. Therefore, $\mathtt{D}_{\mathrm{cr}}\frac{mt}{\mathsf{N}}$ and $\bar{\mathtt{D}}_{\mathrm{cr}}\frac{mt}{\mathsf{N}}$ are the success probability associated with a single non-perfect table and a single perfect DP table, respectively.

Note that Hellman's original matrix stopping rule was $\frac{mt^2}{\mathsf{N}} = 1$ and that the matrix stopping rule used in this work is equivalent to requiring $\bar{\mathtt{D}}_{\mathrm{msc}} \approx 1$. This condition is a rough bound on how large a pre-computation matrix can become before additional pre-computation quickly becomes inefficient in covering more answers and can be understood as a rule for when to stop the creation of a pre-computation matrix.

2.1.2 Non-perfect and Perfect Rainbow Tradeoffs $\mathtt{pR}$, $\mathtt{p}\bar{\mathtt{R}}$

In the following two sections, the exact versions of the rainbow tradeoff are treated in this work will be made more explicit. For any rainbow table variant, one starts by fixing positive integers m and t, satisfying the *matrix stopping rule* $mt \approx \mathsf{N}$, and a small positive integer ℓ. The parameters m, t, and ℓ correspond to the expected number of entries to be stored in each rainbow table, the length of a rainbow pre-computation chain, and the number of tables, respectively. One should note that the matrix stopping rules for the rainbow and DP tradeoffs are different from each other. A *typical environment* for the rainbow tradeoff would call for parameters $m \approx \mathsf{N}^{\frac{2}{3}}$, $t \approx \mathsf{N}^{\frac{1}{3}}$, and a small ℓ, such as 2, or 3, where we allow for a lot of flexibility with the approximations. The rainbow tradeoff requires t reduction functions for each table, so that they are written as $rd_{i,k} : \mathcal{N} \to \mathcal{N}$ ($i = 1, \dots, \ell$, $k = 0, \dots, t-1$). The reduction function $rd_{i,k}$ defines the k-th colored one-way function $F_{i,k} = rd_{i,k} \circ F$ for the i-th rainbow table.

Algorithm 7: Pre-computation phase of $\mathtt{pR}$

```
for $i = 1, \dots, \ell$ do
    $\mathtt{RT}_i \leftarrow \emptyset$;
    for $j = 1, \dots, m$ do
        Choose $\mathtt{sp}^i_j \in \mathcal{N}$;                  // starting point
        $\mathtt{tp} \leftarrow \mathtt{sp}^i_j$;
        for $k = 0, \dots, t-1$ do   // generate pre-computation chain
            $\mathtt{tp} \leftarrow F_{i,k}(\mathtt{tp})$;
        end
        $\mathtt{ep}^i_j \leftarrow \mathtt{tp}$;                       // ending point
        Append $(\mathtt{sp}^i_j, \mathtt{ep}^i_j)$ to $\mathtt{RT}_i$;
    end
    Sort $\mathtt{RT}_i$ according to the $\mathtt{ep}$'s ;
    Record $\mathtt{RT}_i$ to disk;           // non-perfect rainbow table
end
```

The rudimentary forms of the pre-computation and online phases of the non-perfect rainbow tradeoff and those of the perfect rainbow tradeoff [24] are given by Algorithm 7, Algorithm 8, Algorithm 9, and Algorithm 10, respectively.

In the case of the perfect rainbow tradeoff, another positive integer m_0, that needs to be chosen in a manner to be described following, is fixed. The integer m_0 of Algorithm 9 must be set so that each table $\bar{\mathtt{RT}}_i$ is expected to contain m entries. The appropriate value for m_0 is revealed later by Lemma 3.8. As with the DP tradeoff, one could modify the algorithm to incrementally add more starting points until the number of distinct ending points reaches m, but we will not consider such an approach.

Algorithm 8: Online phase of pR

```
for $s = t-1,\dots,0$ do
    for $i = 1,\dots,\ell$ do
        $\mathtt{op}_i \leftarrow rd_{i,s}(\mathbf{h}) = F_{i,s}(\mathbf{p})$;
        for $k = s+1,\dots,t-1$ do
            // generate length-$(t-s)$ online chain
            $\mathtt{op}_i \leftarrow F_{i,k}(\mathtt{op}_i)$;
        end
        Search for $\mathtt{op}_i$ among the ep's of $\bar{\mathtt{RT}}_i$;
        if Matches are found then
            for each pair of $(\mathtt{sp}^i_j, \mathtt{ep}^i_j)$'s in $\mathtt{RT}_i$ with a common $\mathtt{ep}^i_j$==op do
                $\mathtt{tp} \leftarrow \mathtt{sp}^i_j$;
                for $k = 0,\dots,s-1$ do  // regenerate pre-comp chain
                    $\mathtt{tp} \leftarrow F_{i,k}(\mathtt{tp})$;
                end
                if $F(\mathtt{tp})$ == $\mathbf{h}$ then                // answer found
                    return tp as answer and terminate;
                end
            end
        end
    end
end
```

Algorithm 9: Pre-computation phase of $\bar{\mathtt{pR}}$ and $\bar{\mathtt{sR}}$

```
for $i = 1, \dots, \ell$ do
    $\mathtt{RT}_i \leftarrow \emptyset$;
    for $j = 1, \dots, m_0$ do
        Choose $\mathtt{sp}^i_j \in \mathcal{N}$;                        // starting point
        $\mathtt{tp} \leftarrow \mathtt{sp}^i_j$;
        for $k = 0, \dots, t-1$ do    // generate pre-computation chain
            $\mathtt{tp} \leftarrow F_{i,k}(\mathtt{tp})$;
        end
        $\mathtt{ep}^i_j \leftarrow \mathtt{tp}$;                              // ending point
        Append $(\mathtt{sp}^i_j, \mathtt{ep}^i_j)$ to $\mathtt{RT}_i$;
    end
    Sort $\mathtt{RT}_i$ according to the ep's;       // non-perfect rainbow table
    $\bar{\mathtt{RT}}_i \leftarrow \emptyset$;
    for each group of $(\mathtt{sp}, \mathtt{ep})$'s in $\mathtt{RT}_i$ with a common ep do              //
    // remove ep collisions
        Append any one pair to $\bar{\mathtt{RT}}_i$;
    end
    Record $\bar{\mathtt{RT}}_i$ to disk;                  // perfect rainbow table
end
```

Algorithm 10: Online phase of $\bar{\mathtt{pR}}$

```
for $s = t-1, \dots, 0$ do
    for $i = 1, \dots, \ell$ do
        $\mathtt{op}_i \leftarrow rd_{i,s}(\mathbf{h}) = F_{i,s}(\mathbf{p})$;
        for $k = s+1, \dots, t-1$ do
            // generate length-$(t-s)$ online chain
            $\mathtt{op}_i \leftarrow F_{i,k}(\mathtt{op}_i)$;
        end
        Search for $\mathtt{op}_i$ among the ep's of $\bar{\mathtt{RT}}_i$;
        if $\mathtt{op}_i$ == $\mathtt{ep}^i_j$ then
            $\mathtt{tp} \leftarrow \mathtt{sp}^i_j$;
            for $k = 0, \dots, s-1$ do        // regenerate pre-comp chain
                $\mathtt{tp} \leftarrow F_{i,k}(\mathtt{tp})$;
            end
            if $F(\mathtt{tp})$ == $\mathbf{h}$ then                  // answer found
                return $\mathtt{tp}$ as answer and terminate;
            end
        end
    end
end
```

The tables $\mathtt{RT}_i$ produced by Algorithm 7 is called the *i-th non-perfect rainbow table*. The tables $\bar{\mathtt{RT}}_i$ and $\mathtt{RT}_i$ produced by Algorithm 9 are called the *i*-th *perfect rainbow table* and *non-perfect rainbow table*, *corresponding to the perfect rainbow table* $\bar{\mathtt{RT}}$, respectively. As with the DP tradeoff, although the usual definitions of perfect and non-perfect tables cover more general tables, we will deal exclusively with rainbow tables produced by Algorithm 9 in this paper.

Each series of $t+1$ elements spanning from a $\mathtt{sp}^i_j$ to the corresponding $\mathtt{ep}^i_j$ is a *pre-computation rainbow chain*, and the first part of Algorithm 8 and Algorithm 10 generates a *length-*$(t-s)$ *online rainbow chain*. Note that, as with the DP tradeoff, we are following the convention that an online chain starts from the unknown answer $\mathbf{p}$, rather than from $\mathbf{h}$ or $rd_{i,s}(\mathbf{h})$. An online rainbow chain of length $(t-s)$ for the *i*-th rainbow table must start with an application of $F_{i,s}$ and end with the application of $F_{i,t-1}$.

The collection of all pre-computation chains corresponding to $\mathtt{RT}_i$, produced by Algorithm 7, is the *non-perfect rainbow matrix* $\mathtt{RM}_i$. The collection of all pre-computation chains corresponding to $\bar{\mathtt{RT}}_i$, produced by Algorithm 9, is the *perfect rainbow matrix* $\bar{\mathtt{RM}}_i$. This is expected to contain m chains, each of length t, with none of these merging into each other. Likewise, the collection of m_0 chains corresponding to $\mathtt{RT}_i$, produced by Algorithm 9, is the *non-perfect rainbow matrix* $\mathtt{RM}_i$ *corresponding to the perfect rainbow matrix* $\bar{\mathtt{RM}}_i$. As with the DP tradeoff, the reader should be careful to distinguish a rainbow table from a rainbow matrix in reading this paper.

Unlike the DP tradeoff case, pre-computation chains of the rainbow tradeoff are identical in their lengths, and the method of choosing which chain to retain during the ending point collision removal process is irrelevant to our analysis and algorithm performance. Thus, the collision removal [24] process of Algorithm 9 does not specify for any specific method to be used in selecting the chain to be retained in the perfect matrix.

Those not familiar with the DP and rainbow tradeoffs should note that there are big differences between their online phases. First of all, each iteration of the outermost loop appearing in the rainbow tradeoff online phase creates a new online chain for each table, so that t online chains could be created for each table in the worst case. This is in contrast with the DP online phase which creates just one online chain for each pre-computation table. The second significant difference is in the order of table treatment. The DP online phase firstly choose method to treat the pre-computation tables, between serial or parallel manner, but the rainbow online phase treats the small number of tables in parallel [24]. During our theoretical analysis, we make the further simplification that all ℓ tables are processed in parallel, during each iteration of the outermost loop. The final large difference concerns the length of the regenerated pre-computation chain. As with the DP tradeoff, *chain merges* lead to *false alarms* during the online phase of the rainbow tradeoff. To resolve an alarm, the DP tradeoff treated in this paper, which uses the online chain record technique, regenerates the pre-computation chain up to the point of chain merge. On the other hand, the rainbow tradeoff regenerates the pre-computation chain to a a length that is pre-determined by the length of the online chain.

The notation $\mathtt{pR}_{\mathrm{msc}} = \frac{mt}{\mathsf{N}}$ will be used for the non-perfect rainbow tradeoff and this will be refer to as *matrix stopping constant*. The notation $\mathtt{p\bar{R}}_{\mathrm{msc}} = \frac{mt}{\mathsf{N}}$ will be used for the perfect rainbow tradeoff and in this case, the non-perfect rainbow tradeoff analog of $\mathtt{pR}_{\mathrm{msc}} = \frac{m_0 t}{\mathsf{N}}$ will also be used.

2.1.3 Perfect Rainbow Tradeoff, Used in Practice $\mathtt{s\bar{R}}$

Let us now describe another version of the perfect rainbow tradeoff algorithm that would seem the most reasonable way to implement the rainbow tradeoff idea. The algorithm will be referred to as the $\mathtt{s\bar{R}}$ (serial rainbow) tradeoff in this book.

Each perfect rainbow matrix consists of m non-merging pre-computation chains of length t. A total of ℓ pre-computation tables are prepared during the pre-computation phase, as was Algorithm 9. As in the parallel version of the perfect rainbow tradeoff $\bar{\mathtt{pR}}$, a notation $\bar{\mathtt{sR}}_{\text{msc}} = \frac{mt}{\mathsf{N}}$ will also be used to refer the matrix stopping constant of the $\bar{\mathtt{sR}}$ tradeoff, assumed to be not too close to either 0 or 2 in the sense as explanation behind Lemma 3.8.

Algorithm 11: Online phase of $\bar{\mathtt{sR}}$

```
for $i = 1, \dots, \ell$ do
    for $s = t-1, \dots, 0$ do    // generate all online chains at once
        $\mathtt{op}_{t-s} \leftarrow rd_{i,s}(\mathbf{h}) = F_{i,s}(\mathbf{p})$;
        for $k = s+1, \dots, t-1$ do        // length-$(t-s)$ online chain
            $\mathtt{op}_{t-s} \leftarrow F_{i,k}(\mathtt{op}_{t-s})$;
        end
    end
    for $j = 1, \dots, m$ do
        for $s = 1 \dots, t$ do
            if $\mathtt{op}_s$ == $\mathtt{ep}^i_j$ then
                $\mathtt{tp} \leftarrow \mathtt{sp}^i_j$;
                for $k = 0, \dots, t-s-1$ do
                    // regenerate pre-comp chain
                    $\mathtt{tp} \leftarrow F_{i,k}(\mathtt{tp})$;
                end
                if $F(\mathtt{tp})$ == h then                // answer found
                    return tp as answer and terminate;
                end
            end
        end
    end
end
```

Details of the online phase algorithm for the $\bar{\mathtt{sR}}$ tradeoff are given by Algorithm 11. In short, the multiple pre-computation tables are processed serially, one after another, and all t online chains for any one pre-computation table are generated at once before any table searches are performed.

In practice, each pre-computation table is divided into sub-tables that can be fully loaded into fast system memory and the online chains are checked for collisions against these sub-tables rather than against one pre-computation chain at a time. This is the algorithm referred to as PrRb (practical rainbow) and analyzed in [21]. As was discussed there, it is advisable to divide each table into sufficiently many sub-tables, and, in such a case, the theoretically analyzed behavior of Algorithm 11 is very close to that of the sub-tables version. We choose to work with Algorithm 11, which may be viewed as having divided each table into m sub-tables, since it is more suitable for theoretical discussions. To the best of our knowledge, the rainbow tradeoff algorithms implemented by both the RainbowCrack Project [3] and the online phase program `rcracki_mt` [4] for use with the tables from Free Rainbow Tables [1] are essentially this algorithm.

Assuming the pre-computation table is much larger than what can be loaded into the system memory, the approach taken by $\mathtt{s\bar{R}}$ may be considered as being efficient in that no part of the pre-computation table is loaded into memory more than once. In fact, since loading tables from disk to system memory is such a slow process, most programmers will immediately reject the suggestion to interleave the generation of online chains with searches to the very large tables, as is done by Algorithm 10, the online phase of the $\mathtt{p\bar{R}}$ tradeoff.

2.1.4 Other Conventions and Comments

To reduce confusion, in this work, the word *efficiency* is always associated with an algorithm's competitiveness in the use of the online resources, whereas the ability to balance the online efficiency, the pre-computation cost, and sometimes also the success rate, against each other, is referred to with the word *performance*.

The approximation $(1-\frac{1}{b})^a \approx e^{-\frac{a}{b}}$, which is valid when $a = O(b)$, is used frequently throughout this work without any explanation. A more precise statement of this approximation may be found in [18, Appendix A]. Infinite sums are also frequently approximated by appropriate definite integrals throughout this thesis. Both kinds of approximations will be very accurate whenever we use them, as long as a reasonable set of parameters is used with the tradeoff algorithm, and they will be written as equalities rather than as approximations.

For both the DP and rainbow tradeoffs, the parameter t is very roughly of $\mathsf{N}^{\frac{1}{3}}$

order in any practical situation, and we will often ignore approximations that are of $1+O\left(\frac{1}{t}\right)$ order multiplicative factors and write these as equalities.

Applications of the perfect table technique to the DP and rainbow tradeoffs are expected to increase both the online efficiency and the pre-computation cost. Hence, it is not clear if the benefits of using perfect tables and its parallel treatment outweigh its drawback. Providing information that can be used to settle this question is one of the objectives of this work. Truncation of ending points must also be used carefully, since the storage reduction is associated with an increase in online time. However, all other techniques we are employing are only advantageous, when used appropriately in any practical situation. Details of those techniques we applied are discussed in the next section.

2.2 Storage Optimization Techniques

There are some known techniques to reduce the number of bits required to store a single table entry of pre-computation table. In this section, the techniques taken into account, when analyzing the DP and rainbow variants, that are clarified in Section 2.1.1 and Section 2.1.2.

Let us consider about the DP variants. Firstly, the starting points $\{\mathtt{sp}^i_j\}_{j=1}^{\vec{m}_0}$ for a pre-computation matrix must be chosen to be distinct. We specify more concretely that, within each table, sequential starting points [6, 9, 10] are to be used. Then, each starting point can be recorded in $\log \vec{m}_0$ bits, which should be much smaller than the $\log \mathsf{N}$ bits required to record a random element of $\mathcal{N}$.

The fact that every ending point satisfies the distinguishing property implies that certain parts of the ending points are redundant. These parts are not recorded to the pre-computation table to save $\log t$ bits of storage per ending point [9]. In addition, the ending points are further truncated to a certain length before being written to storage [8, 9]. Since some ending point information is lost by the truncation, only probable matches can then be announced during the online phase, and this will increase the frequency of false alarms. However, this side effect of truncation can be maintained at a manageable level by controlling the degree of truncation. When dealing with non-perfect DP tables, it is verified that close to $\log t$ bits from each ending point can be truncated with minimal effect on the online running time [18]. Details for the perfect DP case are discussed in this work.

Note that, since the pre-computation table is sorted on the ending points, some of the most significant bits of the ending points will be increasing almost predictably throughout the table. This observation is the basis of the index file technique [9], which allows the removal of close to $\log m$ further bits of storage per truncated ending point without any loss of ending point information. We assume that this storage reduction technique is also used in recording the pre-computation tables.

Techniques for reducing the number of bits allocated to each table entry of the rainbow tradeoff that will be considered in this paper are all of those that were previously explained for the DP tradeoff, except for the one involving the definition of a DP. Sequential starting points [6, 9, 10] are used, so that only $\log m$ or $\log m_0$ bits for non-perfect or perfect table are needed to record each starting point, respectively. Ending points of non-perfect rainbow tables can be truncated so that slightly more than $\log m$ bits remain without visible side-effects on the online running time [18]. Details for the perfect rainbow case are also discussed in this work. Finally, the index file technique [8, 9] is also applied, so that about $\log m$ further bits per each truncated ending point can be removed with no loss of information.

2.3 Previous Results

The original DP and non-perfect rainbow tradeoffs were analyzed in [18]. As a its following work, the parallel DP tradeoff was also analyzed in [16]. In this section, we quickly review results from [18] and [16] that are required in this paper and introduce some more notation.

2.3.1 Analyses of the Original DP and Parallel DP Tradeoffs

In order to produce a single non-perfect DP table, having m non-necessarily distinct entries, $\vec{m}_0 \approx m$ starting distinct starting points are needed, when sufficiently large chain length bound $\hat{t}$ is used, which we already assumed at the algorithm clarification in Section 2.1.1.

As defined previously, given a non-perfect DP matrix, its coverage rate is defined to be the number of distinct nodes that appear among the DP chains as inputs to the one-way function F, divided by mt. Note that the DPs ending each pre-computation chain are not counted in this definition. The expected coverage rate of a non-perfect DP matrix generated from $\vec{m}_0 \approx m$ random distinct starting points can be computed through the formula

$$\mathtt{D}_{\text{cr}} = \frac{2}{\sqrt{1+2\mathtt{D}_{\text{msc}}}+1}, \tag{2.1}$$

where $\mathtt{D}_{\text{msc}} = \frac{mt^2}{\mathsf{N}}$. In particular, the coverage rate can be seen as a function of the single variable $\mathtt{D}_{\text{msc}}$, rather than the separate parameters m, t, and N.

Let $\mathtt{D}_{\text{pc}} = \frac{mt\ell}{\mathsf{N}}$ be the pre-computation coefficient so that $\mathtt{D}_{\text{pc}}\mathsf{N}$ is the pre-computation cost. It is not difficult to show that the probability of success for the DP tradeoff can be expressed as $\mathtt{D}_{\text{ps}} = 1 - e^{-\mathtt{D}_{\text{pc}}\mathtt{D}_{\text{cr}}}$. If we rewrite this equation in the form

$$\mathtt{D}_{\text{pc}} = -\frac{\ln(1-\mathtt{D}_{\text{ps}})}{\mathtt{D}_{\text{cr}}} = -\frac{\ln(1-\mathtt{D}_{\text{ps}})}{2}\left(\sqrt{1+2\mathtt{D}_{\text{msc}}}+1\right), \tag{2.2}$$

we can see that, under a fixed requirement on the success rate of the DP tradeoff, the pre-computation coefficient $\mathtt{D}_{\text{pc}}$ is a function of the matrix stopping constant $\mathtt{D}_{\text{msc}}$. Remark that, for identical matrix stopping constant $\mathtt{D}_{\text{msc}} = \frac{mt^2}{\mathsf{N}} = \mathrm{pD}_{\text{msc}}$, the original DP and parallel DP tradeoff algorithms agree the same coverage rate $\mathtt{D}_{\text{cr}}$, pre-computation coefficient $\mathtt{D}_{\text{pc}}$, and success probability $\mathtt{D}_{\text{ps}}$, since they share the same pre-computation phase which produce non-perfect DP tables by fixed matrix stopping rule. Also, a useful formula is that when visualizing a non-perfect DP matrix as having been aligned at the ending points, the number of distinct points found in a column of distance i from the ending points is expected to be

$$\overset{\leftarrow}{m}_i = \mathtt{D}_{\mathrm{cr}}\, m\Big(1-\frac{1}{t}\Big)^{i-1}. \tag{2.3}$$

Finally, the time memory tradeoff curve for the original and parallel DP tradeoffs are given by $TM^2 = \mathtt{D}_{TM^2}\mathsf{N}^2$ and $TM^2 = \mathtt{pD}_{TM^2}\mathsf{N}^2$, where the tradeoff coefficients are

$$\mathtt{D}_{TM^2} = \Big(2+\frac{1}{\mathtt{D}_{\mathrm{msc}}}\Big)\frac{1}{\mathtt{D}_{\mathrm{cr}}^3}\,\mathtt{D}_{\mathrm{ps}}\big\{\ln(1-\mathtt{D}_{\mathrm{ps}})\big\}^2 \tag{2.4}$$

and

$$\mathtt{pD}_{TM^2} = \Big(\frac{\ln(1-\mathtt{pD}_{\mathrm{ps}})}{\mathtt{pD}_{\mathrm{ps}}}\int_0^1 (1-\mathtt{pD}_{\mathrm{ps}})^{1-u}\ln u\, du + \frac{1}{\mathtt{pD}_{\mathrm{msc}}}\Big)\frac{1}{\mathtt{D}_{\mathrm{cr}}^3}\,\mathtt{pD}_{\mathrm{ps}}\big\{\ln(1-\mathtt{pD}_{\mathrm{ps}})\big\}^2, \tag{2.5}$$

respectively. Note that T is the number of F-iterations to be required during the online phase and M is the number of pre-computation table entries to be stored. When placed under a fixed success rate requirement $\mathtt{D}_{\mathrm{ps}}$, $\mathtt{D}_{TM^2}$ can also be seen as a function of $\mathtt{D}_{\mathrm{msc}}$, through a substitution of (2.1) and so can $\mathtt{pD}_{TM^2}$.

[18] also utilized the storage optimization techniques as explained in Section 2.2 to analyze tradeoff algorithms. As a result, slightly more than $\log m$ bits are enough to store a single table entry $(\mathtt{sp}, \mathtt{ep})$ of a non-perfect DP table.

2.3.2 Analysis of the Non-perfect Rainbow Tradeoff

The pre-computation coefficient $\mathtt{pR}_{\mathrm{pc}}$, which means that $\mathtt{pR}_{\mathrm{pc}}\mathsf{N}$ F-iterations are required during the pre-computation phase, and the success probability for the non-perfect rainbow tradeoff are written as the forms

$$\mathtt{pR}_{\mathrm{pc}} = \mathtt{pR}_{\mathrm{msc}}\ell \tag{2.6}$$

and

$$\mathsf{pR}_{\mathrm{ps}} = 1 - \Big(\frac{2}{2+\mathsf{pR}_{\mathrm{msc}}}\Big)^{2\ell}, \tag{2.7}$$

respectively, where $\mathsf{pR}_{\mathrm{msc}} = \frac{mt}{\mathsf{N}}$ and ℓ is the number of tables.

From (2.6) and (2.7), one can easily see that the pre-computation coefficient can be regarded as a function of the matrix stopping constant $\mathsf{pR}_{\mathrm{msc}}$, when a success probability is fixed.

The time memory tradeoff curve for the non-perfect rainbow tradeoff is given by $TM^2 = \mathsf{pR}_{TM^2}\mathsf{N}^2$, where the tradeoff coefficient is

$$\mathsf{pR}_{TM^2} = \frac{\ell^3}{(2\ell+1)(2\ell+2)(2\ell+3)} \left(\begin{array}{l} \{(2\ell-1)+(2\ell+1)\mathsf{pR}_{\mathrm{msc}}\}(2+\mathsf{pR}_{\mathrm{msc}})^2 \\ -4\{(2\ell-1)+\ell(2\ell+3)\mathsf{pR}_{\mathrm{msc}}\}\Big(\frac{2}{2+\mathsf{pR}_{\mathrm{msc}}}\Big)^{2\ell} \end{array} \right). \tag{2.8}$$

When placed under a fixed success rate requirement $\mathsf{pR}_{\mathrm{ps}}$, this can also be seen as a function of $\mathsf{pR}_{\mathrm{msc}}$ through a substitution of (2.7).

Finally, only slightly more than $\log m$ bits are needed to store a single table entry of a non-perfect rainbow table, as a result of the storage optimization.

Chapter 3

Perfect Table Tradeoff Algorithms

In this chapter, analyses of the perfect DP and perfect rainbow tradeoff will be presented. All of the results were previously published in [22].

3.1 Analysis of the Perfect DP Tradeoff

In this section, we provide a full analysis of the perfect DP tradeoff that uses a sufficiently large upper bound on the chain length. A more clarified description of the perfect DP tradeoff that is being treated in this work was given in Section 2.1.1.

3.1.1 Online Efficiency

We will present formulas describing the success probability, pre-computation cost, and tradeoff coefficient of the perfect DP tradeoff. The discussion will require previous results concerning the non-perfect DP tradeoff.

Let us visualize a non-perfect DP matrix with the ending points aligned in a single column. Some of the rows (pre-computation chains) will be merging into each other. Let us use $\overset{\leftarrow}{m}_k$ to denote the number of distinct points expected in its column that is k iterations away from the ending points in a *non-perfect* DP matrix. In particular, $\overset{\leftarrow}{m}_0$ denotes the number of distinct ending points and this is also the number of independent or non-overlapping rows of the non-perfect DP matrix.

Lemma 3.1. *The number of distinct ending points in a non-perfect DP matrix may be approximated by the number of its distinct points that are a single iteration away from the ending point DPs. More precisely, we have* $\overleftarrow{m}_0 = \overleftarrow{m}_1\{1+O(\frac{1}{t})\}$.

Proof. By the definitions of $\overleftarrow{m}_0$ and $\overleftarrow{m}_1$, $\overleftarrow{m}_0$ can be interpreted as the expected size of F-image produced from $\overleftarrow{m}_1$ inputs. Recall that we are treating F as a random function and note that the set of DPs is of size N/t. Viewing this situation as that of making $\overleftarrow{m}_1$ independent random choices from the set of all DPs, the fraction of the DP space that is not hit by any of the $\overleftarrow{m}_1$ choices becomes $\left(1-\frac{1}{\mathsf{N}/t}\right)^{\overleftarrow{m}_1}$, and the expected size of the image set can be written as

$$\overleftarrow{m}_0 = (\mathsf{N}/t)\left\{1-\left(1-\frac{1}{\mathsf{N}/t}\right)^{\overleftarrow{m}_1}\right\}.$$

After expanding the $\overleftarrow{m}_1$-th power to write

$$\begin{aligned}\overleftarrow{m}_0 &= \frac{\mathsf{N}}{t}\left\{1-1+\frac{\overleftarrow{m}_1 t}{\mathsf{N}}-\binom{\overleftarrow{m}_1}{2}\left(\frac{t}{\mathsf{N}}\right)^2+\binom{\overleftarrow{m}_1}{3}\left(\frac{t}{\mathsf{N}}\right)^3-\cdots\right\}\\ &= \overleftarrow{m}_1-\binom{\overleftarrow{m}_1}{2}\left(\frac{t}{\mathsf{N}}\right)+\binom{\overleftarrow{m}_1}{3}\left(\frac{t}{\mathsf{N}}\right)^2-\cdots,\end{aligned}$$

we can recall the condition $mt^2 \approx \mathsf{N}$ and note $\overleftarrow{m}_1 = \Theta(m)$ to observe $\frac{\overleftarrow{m}_1 t}{\mathsf{N}} \ll 1$ and claim

$$\overleftarrow{m}_0 = \overleftarrow{m}_1 + \overleftarrow{m}_1 O\left(\frac{\overleftarrow{m}_1 t}{\mathsf{N}}\right) = \overleftarrow{m}_1\left\{1+O\left(\frac{1}{t}\right)\right\}.$$

Thus, we may approximate $\overleftarrow{m}_0$ with $\overleftarrow{m}_1$, for any realistic value of t. In fact, we had explicitly stated in Section 2.1.4 that any approximation of $1+O\left(\frac{1}{t}\right)$ order multiplicative factor would be ignored and written as an equality. □

Recalling the previous results, presented in Section 2.3.1, we can rewrite (2.3) as

$$\overleftarrow{m}_k = |\mathtt{DM}|\left(1-\frac{1}{t}\right)^{k-1}\frac{1}{t}, \tag{3.1}$$

for $k \geq 1$. Here, the $|\mathtt{DM}|$ denotes the number of distinct points expected in a non-perfect DP matrix, as defined before. To be more precise, the $|\mathtt{DM}|$ used here counts the points that were used as inputs to the iterating function during the non-perfect DP table creation, so that the starting points are included and the ending points are excluded.

It is also already known from (2.1) that a single non-perfect DP matrix created with $\vec{m}_0$ starting points is expected to contain

$$|\mathtt{DM}| = \frac{2\vec{m}_0 t}{1+\sqrt{1+2\mathtt{D}_{\mathrm{msc}}}} \tag{3.2}$$

distinct points, where $\mathtt{D}_{\mathrm{msc}} = \frac{\vec{m}_0 t^2}{\mathsf{N}}$ is the matrix stopping constant for the corresponding non-perfect DP matrix. The reader should be careful to distinguish the symbol $\vec{m}_0$ from the previously used symbol $\overleftarrow{m}_0$. The information, which has been obtained, will be used to find out relations between a perfect DP matrix and its corresponding non-perfect DP matrix. As a beginning of those, the expected number of starting points required to produce a perfect DP matrix which consists of m entries.

Lemma 3.2. *A non-perfect DP matrix created with $\vec{m}_0$ randomly generated starting points is expected to contain $\frac{2\vec{m}_0}{1+\sqrt{1+2\mathtt{D}_{\mathrm{msc}}}}$ distinct ending points, where $\mathtt{D}_{\mathrm{msc}} = \frac{\vec{m}_0 t^2}{\mathsf{N}}$. Conversely, given m, one must generate $\vec{m}_0 = \left(1+\frac{\bar{\mathtt{D}}_{\mathrm{msc}}}{2}\right)m$ chains, where $\bar{\mathtt{D}}_{\mathrm{msc}} = \frac{mt^2}{\mathsf{N}}$, in order for m to be the expected number of chains contained in the corresponding perfect DP matrix.*

Proof. Ignoring $1+O(\frac{1}{t})$ multiplicative factor of the statement in Lemma 3.1, we may even write $\overleftarrow{m}_0 = \overleftarrow{m}_1$. Recalling from (3.1) that $\overleftarrow{m}_1 = |\mathtt{DM}|\frac{1}{t}$ and combining this with (3.2), we arrive at

$$\overleftarrow{m}_0 = \overleftarrow{m}_1 = \frac{|\mathtt{DM}|}{t} = \frac{2\vec{m}_0}{1+\sqrt{1+2\vec{m}_0 t^2/\mathsf{N}}},$$

which is the first claim.

As for the second claim, given m, replacing $\overleftarrow{m}_0$ of the above relation by m, it suffices to solve for $\overrightarrow{m}_0$ from the equation

$$m = \frac{2\overrightarrow{m}_0}{1+\sqrt{1+2\overrightarrow{m}_0 t^2/\mathsf{N}}}.$$

Recalling the notation $\bar{\mathtt{D}}_{\mathrm{msc}} = \frac{mt^2}{\mathsf{N}}$, we can rewrite this in the form

$$1+\sqrt{1+2\bar{\mathtt{D}}_{\mathrm{msc}}\frac{\overrightarrow{m}_0}{m}} = 2\,\frac{\overrightarrow{m}_0}{m}$$

and again into the form

$$1+2\bar{\mathtt{D}}_{\mathrm{msc}}\frac{\overrightarrow{m}_0}{m} = \Big(2\,\frac{\overrightarrow{m}_0}{m}-1\Big)^2.$$

Solving this quadratic equation in $\frac{\overleftarrow{m}_0}{m}$ and discarding the meaningless solution $\frac{\overleftarrow{m}_0}{m} = 0$, we find

$$\frac{\overrightarrow{m}_0}{m} = 1+\frac{\bar{\mathtt{D}}_{\mathrm{msc}}}{2},$$

which is the second claim. □

Note that the first claim of this lemma gives a simple formula to express the number of chains remaining after the removal of merges, which many previous works [27] had attempted to find.

Throughout this thesis,

$$\overrightarrow{m}_0 = \Big(1+\frac{\bar{\mathtt{D}}_{\mathrm{msc}}}{2}\Big)m \tag{3.3}$$

will always denote the number of starting points that are required to create a perfect DP table that is expected to contain m ending points. This is the value of $\overrightarrow{m}_0$ that should be used by Algorithm 1, given the algorithm parameters m and t.

By multiplying $\frac{t^2}{\mathsf{N}}$ to both sides and recalling the definitions $\mathtt{D}_{\mathrm{msc}} = \frac{\overrightarrow{m}_0 t^2}{\mathsf{N}}$ and

$\bar{\mathtt{D}}_{\text{msc}} = \frac{mt^2}{\mathsf{N}}$, we can rewrite the above as

$$\mathtt{D}_{\text{msc}} = \Big(1 + \frac{\bar{\mathtt{D}}_{\text{msc}}}{2}\Big)\bar{\mathtt{D}}_{\text{msc}}.$$

Viewing this as a quadratic equation concerning the indeterminate $\bar{\mathtt{D}}_{\text{msc}}$, we can solve for $\bar{\mathtt{D}}_{\text{msc}}$ to obtain

$$\bar{\mathtt{D}}_{\text{msc}} = \sqrt{1 + 2\mathtt{D}_{\text{msc}}} - 1. \tag{3.4}$$

This can be used to convert any formula given in terms of $\bar{\mathtt{D}}_{\text{msc}}$ into one given in terms of $\mathtt{D}_{\text{msc}}$. Also one can state that (3.4) presents the relation between a perfect DP matrix and its corresponding non-perfect DP matrix.

In addition, when a DP matrix is created from $\vec{m}_0$ starting points, where $\vec{m}_0$ is as given by (3.3), it contains

$$|\mathtt{DM}| = mt \tag{3.5}$$

distinct points, used as the inputs to the one-way function. That is, given the perfect table parameters m and t, the corresponding non-perfect DP matrix, which must be created from $\vec{m}_0$ starting points with $\vec{m}_0$ as given by (3.3), is expected to cover mt distinct points.

Note that by combining (3.1) and (3.5), we obtain the useful formula

$$\overleftarrow{m}_k = m\Big(1 - \frac{1}{t}\Big)^{k-1}. \tag{3.6}$$

The pre-computation phase of a perfect DP tradeoff requires $\vec{m}_0 t\ell$ iterations of the one-way function. We define the *pre-computation coefficient* for the perfect DP tradeoff to be $\bar{\mathtt{D}}_{\text{pc}} = \frac{\vec{m}_0 t\ell}{\mathsf{N}}$, so that the cost of pre-computation becomes $\bar{\mathtt{D}}_{\text{pc}}\mathsf{N}$. The following statement is a direct consequence of Lemma 3.2 or (3.3).

Proposition 3.1. *The pre-computation coefficient of the perfect DP tradeoff is*

$$\bar{\mathtt{D}}_{\text{pc}} = \Big(1 + \frac{\bar{\mathtt{D}}_{\text{msc}}}{2}\Big)\frac{mt\ell}{\mathsf{N}}.$$

By the definition of the coverage rate, which was given at the end of Section 2.1.1, a single perfect DP matrix contains the correct answer **p** to the given inversion target $\mathbf{h} = F(\mathbf{p})$ with probability $\frac{mt\bar{\mathtt{D}}_{\mathrm{cr}}}{\mathsf{N}}$. Thus, the success probability of the complete perfect DP tradeoff may be stated as

$$\bar{\mathtt{D}}_{\mathrm{ps}} = 1 - \left(1 - \frac{mt\,\bar{\mathtt{D}}_{\mathrm{cr}}}{\mathsf{N}}\right)^{\ell} = 1 - \exp\left(-\frac{mt\ell}{\mathsf{N}}\bar{\mathtt{D}}_{\mathrm{cr}}\right), \tag{3.7}$$

where we are relying on the approximation stated in Section 2.1.4 for the second equality, and we can combine this with Proposition 3.1 to claim the following.

Proposition 3.2. *The success probability of the perfect DP tradeoff is*

$$\bar{\mathtt{D}}_{\mathrm{ps}} = 1 - \exp\left(-\frac{2\,\bar{\mathtt{D}}_{\mathrm{pc}}\,\bar{\mathtt{D}}_{\mathrm{cr}}}{2+\bar{\mathtt{D}}_{\mathrm{msc}}}\right).$$

We have computed expressions for $\bar{\mathtt{D}}_{\mathrm{pc}}$ and $\bar{\mathtt{D}}_{\mathrm{ps}}$ that do not involve $\vec{m}_0$. Some technical lemmas need to be prepared first to obtain such an expression for $\bar{\mathtt{D}}_{\mathrm{cr}}$.

Given a function $F : \mathcal{N} \to \mathcal{N}$ and a non-negative integer k, we define $\mathscr{D}_k(F)$ or $\mathscr{D}_k$ to be the set of elements of $\mathcal{N}$ that are k-many F-iterations away from their closest DPs. In particular, $\mathscr{D}_0$ is the set of DPs. It is clear that $\{\mathscr{D}_k(F)\}_{k=0}^{\infty}$ becomes a partition of $\mathcal{N}$, and that we can expect the sizes of these subsets to be

$$|\mathscr{D}_k| = \mathsf{N}\left(1 - \frac{1}{t}\right)^k \frac{1}{t}, \tag{3.8}$$

for a random function. Note that the above argument ignores the possibility of encountering loops, but this can be justified since $mt^2 \approx \mathsf{N}$ implies $t \ll \sqrt{\mathsf{N}}$ and the rho length of a random walk initiated from a random point is expected to be of $\sqrt{\mathsf{N}}$ order.

Lemma 3.3. *Let $F : \mathcal{N} \to \mathcal{N}$ be chosen uniformly at random from the set of all functions acting on $\mathcal{N}$ and let us fix a set $D \subset \mathscr{D}_k(F)$ for some $k \geq 1$. Then the* *expect sizes of its iterated images under F will satisfy*

$$\frac{|F^i(D)|}{\mathsf{N}(\left(1-\frac{1}{t}\right))^{k-i}\frac{1}{t}} = 1-\exp\left(-\frac{|F^{i-1}(D)|}{\mathsf{N}(\left(1-\frac{1}{t}\right))^{k-i}\frac{1}{t}}\right),$$

for each $i = 1,\dots,k$.

Proof. As a trivial generalizing of the argument in the proof of Lemma 3.1, for a random function $F : \mathscr{A} \to \mathscr{B}$ defined on finite sets and a subset $\mathscr{C}$ of the domain $\mathscr{A}$, the image size of $\mathscr{C}$ is expected to be

$$|F(\mathscr{C})| = |\mathscr{B}|\Big\{1-\Big(1-\frac{1}{|\mathscr{B}|}\Big)^{|\mathscr{C}|}\Big\} = |\mathscr{B}|\Big\{1-\exp\Big(-\frac{|\mathscr{C}|}{|\mathscr{B}|}\Big)\Big\},$$

assuming $|\mathscr{C}| = O(|\mathscr{B}|)$. The claim is now a direct consequence of the set sizes given by (3.8). Note that, since $|\mathscr{D}_j| \le |\mathscr{D}_{j-1}|$, we need not worry about the $|\mathscr{C}| = O(|\mathscr{B}|)$ condition. □

It is possible to work out the iterations expressed by this lemma and write down each iterated image size as a closed-form formula.

Lemma 3.4. *Let $F : \mathscr{N} \to \mathscr{N}$ be a random function and let $D \subset \mathscr{D}_k(F)$, for some $k \ge 0$. When $|D| = O(m)$, the size of the i-th iterated image of D under F is expected to be*

$$|F^i(D)| = \frac{2|D|}{2+\bar{\mathsf{D}}_{\mathrm{msc}}\frac{|D|}{m}e^{\frac{k}{t}}(1-e^{-\frac{i}{t}})},$$

for each $0 \le i \le k$.

Proof. Let us temporarily introduce the notation $f_i = \frac{|F^i(D)|}{\mathsf{N}(1-\frac{1}{t})^{k-i}\frac{1}{t}}$, and rewrite Lemma 3.3 as

$$f_i = 1-\exp\Big\{-\Big(1-\frac{1}{t}\Big)f_{i-1}\Big\} = \Big(1-\frac{1}{t}\Big)f_{i-1} - \frac{1}{2}\Big(1-\frac{1}{t}\Big)^2 f_{i-1}^2 + \cdots.$$

The condition $|D| = O(m)$ implies $|F^i(D)| = O(m)$, so that $f_i = O\big(\frac{m}{\mathsf{N}/t}\big) = O\big(\frac{1}{t}\big)$, and we can state

$$f_i - f_{i-1} = -\frac{1}{t} f_{i-1} - \frac{1}{2} f_{i-1}^2 + O\Big(\frac{f_{i-1}^2}{t}\Big).$$

Noting that $\frac{f_{i-1}^2}{t}$ is of strictly smaller order than $\frac{f_{i-1}}{t} + \frac{f_{i-1}^2}{2}$, we can ignore the final term. Recall that the Euler method allows for the solution of a ordinary differential equation with a given initial value to be approximately expressed as an iterative sequence. Applying this in the reverse direction, we can solve the differential equation

$$f'(x) = -\frac{1}{t} f(x) - \frac{1}{2} f(x)^2$$

associated with the above difference equation, with the initial condition $f(0) = f_0 = \frac{|D|t}{\mathsf{N}e^{-\frac{k}{t}}}$, to obtain

$$f_i = \frac{2|D|t}{2\mathsf{N}e^{\frac{i-k}{t}} + (e^{\frac{i}{t}} - 1)|D|t^2}.$$

Recalling the definition of f_i, we can state

$$|F^i(D)| = \frac{2\mathsf{N}(\big(1 - \frac{1}{t}\big))^{k-i}|D|}{2\mathsf{N}e^{\frac{i-k}{t}} + (e^{\frac{i}{t}} - 1)|D|t^2} = \frac{2\mathsf{N}e^{\frac{i-k}{t}}|D|}{2\mathsf{N}e^{\frac{i-k}{t}} + (e^{\frac{i}{t}} - 1)|D|t^2},$$

and a direct simplification of this equation, using the notation $\bar{\mathsf{D}}_{\text{msc}} = \frac{mt^2}{\mathsf{N}}$, results in our claim. □

The previous two lemmas were prepared to support the next lemma, which gives the probability for a single chain not to merge into a set of chains. This information will be used to derive the coverage rate $\bar{\mathsf{D}}_{\text{cr}}$ of a perfect DP matrix.

Lemma 3.5. *Let $F : \mathcal{N} \to \mathcal{N}$ be a random function and let $D \subset \mathcal{D}_k(F)$, for some k. When $|D| = O(m)$, the probability for a random point $x \in \mathcal{D}_k(F)$ to satisfy $F^k(x) \notin F^k(D)$ is*

$$\Big\{1 + \frac{\bar{\mathsf{D}}_{\text{msc}}}{2} \frac{|D|}{m} \big(e^{\frac{k}{t}} - 1\big)\Big\}^{-2}.$$

Proof. Since the starting point itself and each subsequent iterations of the random function must not contain in the iterated image sets, the probability in question is

$$\prod_{i=0}^{k}\Big(1-\frac{|F^i(D)|}{|\mathscr{D}_{k-i}|}\Big)=\prod_{i=0}^{k}\Big(1-\frac{|F^i(D)|}{\mathsf{N}(1-\frac{1}{t})^{k-i}\frac{1}{t}}\Big)=\Big(1-\frac{|D|t}{\mathsf{N}e^{-\frac{k}{t}}}\Big)\prod_{i=1}^{k}\Big(1-\frac{|F^i(D)|}{\mathsf{N}(1-\frac{1}{t})^{k-i}\frac{1}{t}}\Big).$$

Here, the first equality is based on (3.8). By applying Lemma 3.3 to the product of k terms, we can write

$$\begin{aligned}\prod_{i=1}^{k}\Big(1-\frac{|F^i(D)|}{\mathsf{N}(1-\frac{1}{t})^{k-i}\frac{1}{t}}\Big)&=\prod_{i=1}^{k}\exp\Big(-\frac{|F^{i-1}(D)|}{\mathsf{N}(1-\frac{1}{t})^{k-i}\frac{1}{t}}\Big)\\&=\exp\left(-\Big(1-\frac{1}{t}\Big)\sum_{i=0}^{k-1}\frac{|F^i(D)|}{\mathsf{N}(1-\frac{1}{t})^{k-i}\frac{1}{t}}\right).\end{aligned}$$

Since we are given the condition $|D|=O(m)$, we can apply Lemma 3.4, or the last equation in its proof, and compute the sum inside the exponential function as

$$\sum_{i=0}^{k-1}\frac{|F^i(D)|}{\mathsf{N}\big(1-\frac{1}{t}\big)^{k-i}\frac{1}{t}}=\sum_{i=0}^{k-1}\frac{2|D|t}{2\mathsf{N}e^{\frac{i-k}{t}}+(e^{\frac{i}{t}}-1)|D|t^2}=\sum_{i=0}^{k-1}\frac{2|D|t}{\frac{2\mathsf{N}}{t}e^{\frac{i-k}{t}}+(e^{\frac{i}{t}}-1)|D|t}\frac{1}{t}.$$

Viewing this as the left Riemann sum of the function $\Phi(u):=\frac{2|D|t}{\frac{2\mathsf{N}}{t}e^{u-\frac{k}{t}}+(e^u-1)|D|t}$ on the interval $[0,\frac{k}{t}]$, we can approximate this with the definite integral

$$\int_0^{k/t}\frac{2|D|t}{\frac{2\mathsf{N}}{t}e^{-\frac{k}{t}}e^u+(e^u-1)|D|t}\,du=2\ln\Big\{1+\frac{|D|t^2}{2\mathsf{N}}\big(e^{\frac{k}{t}}-1\big)\Big\}.$$

By substituting the sum back into the exponential function, we get

$$\begin{aligned}\prod_{i=0}^{k}\Big(1-\frac{|F^i(D)|}{|\mathscr{D}_{k-i}|}\Big)&=\Big(1-\frac{|D|t}{\mathsf{N}e^{-\frac{k}{t}}}\Big)\exp\Big(-\Big(1-\frac{1}{t}\Big)2\ln\Big\{1+\frac{|D|t^2}{2\mathsf{N}}\big(e^{\frac{k}{t}}-1\big)\Big\}\Big)\\&=\Big(1-\frac{|D|t}{\mathsf{N}e^{-\frac{k}{t}}}\Big)\Big\{1+\frac{\bar{\mathsf{D}}_{\mathrm{msc}}}{2}\frac{|D|}{m}\big(e^{\frac{k}{t}}-1\big)\Big\}^{-2(1-\frac{1}{t})}.\end{aligned}$$

The $\big(1-\frac{1}{t}\big)$ term appearing in the exponent is insignificant and the condition

$|D| = O(m)$ allows us to ignore the first product term, which is of $1 - O\left(\frac{1}{t}\right)$ order. We have arrived at the claimed formula. □

With the help of the technical lemmas that have been obtained, we can finally present the coverage rate of a perfect DP matrix.

Proposition 3.3. *The coverage rate of a perfect DP matrix is*

$$\bar{\mathtt{D}}_{\mathrm{cr}} = \frac{2}{\bar{\mathtt{D}}_{\mathrm{msc}}} \ln\left(1 + \frac{\bar{\mathtt{D}}_{\mathrm{msc}}}{2}\right).$$

Proof. Let us consider a non-perfect DP matrix before the chain removal process to produce a perfect DP matrix. A chain of the non-perfect DP matrix survives through the collision removal process if and only if it does not merge into another chain of length longer than or equal to its length. To be more precise, according to Lemma 3.5, the probability for a chain of length k in a non-perfect DP matrix to remain in the perfect matrix is

$$\left\{1 + \frac{\bar{\mathtt{D}}_{\mathrm{msc}}}{2} \frac{\overleftarrow{m}_k}{m} \left(e^{\frac{k}{t}} - 1\right)\right\}^{-2}.$$

The expected number of chains that are of length k is $\vec{m}_0\left(1 - \frac{1}{t}\right)^{k-1}\frac{1}{t}$, before the removal of merges, and each chain of length k contains information of k points, used as inputs to the one-way function. Since chains in a perfect DP matrix do not have common points with each other, the number of distinct points in the perfect DP table except DPs is

$$\sum_{k=1}^{\infty} k \cdot \vec{m}_0 \left(1 - \frac{1}{t}\right)^{k-1} \frac{1}{t} \cdot \left\{1 + \frac{\bar{\mathtt{D}}_{\mathrm{msc}}}{2} \frac{\overleftarrow{m}_k}{m} \left(e^{\frac{k}{t}} - 1\right)\right\}^{-2}.$$

The coverage rate of the perfect DP matrix can thus be given by

$$\bar{\mathtt{D}}_{\mathrm{cr}} = \frac{1}{mt} \vec{m}_0 \left(1 - \frac{1}{t}\right)^{-1} \sum_{k=1}^{\infty} \frac{k}{t} \cdot e^{-\frac{k}{t}} \cdot \left\{1 + \frac{\bar{\mathtt{D}}_{\mathrm{msc}}}{2} e^{-\frac{k}{t}} \left(1 - \frac{1}{t}\right)^{-1} \left(e^{\frac{k}{t}} - 1\right)\right\}^{-2},$$

where we have used (3.6) to remove the $\overleftarrow{m}_k$ term. After ignoring the insignificant $(1-\frac{1}{t})^{-1}$ terms, we rewrite the above as

$$\bar{\mathtt{D}}_{\mathrm{cr}} = \frac{\vec{m}_0}{m} \sum_{k=1}^{\infty} \frac{k}{t} \cdot e^{-\frac{k}{t}} \cdot \Big\{ 1 + \frac{\bar{\mathtt{D}}_{\mathrm{msc}}}{2} \big(1 - e^{-\frac{k}{t}}\big) \Big\}^{-2} \frac{1}{t}$$

and interpret this as a definite integral to compute the coverage rate as

$$\bar{\mathtt{D}}_{\mathrm{cr}} = \frac{\vec{m}_0}{m} \int_0^{\infty} u e^{-u} \Big\{ 1 + \frac{\bar{\mathtt{D}}_{\mathrm{msc}}}{2} \big(1 - e^{-u}\big) \Big\}^{-2} du = \frac{\vec{m}_0}{m} \frac{\ln\big(1 + \frac{\bar{\mathtt{D}}_{\mathrm{msc}}}{2}\big)}{\frac{\bar{\mathtt{D}}_{\mathrm{msc}}}{2}\big(1 + \frac{\bar{\mathtt{D}}_{\mathrm{msc}}}{2}\big)}.$$

It now suffices to recall Lemma 3.2 or (3.3) to arrive at the claimed formula. □

Let us briefly discuss the average chain length of a perfect DP matrix. By definition, it is the number of distinct points in a perfect DP matrix divided by the number of its distinct ending points, and according to the above lemma, it can be written as a formula

$$\frac{|\bar{\mathtt{DM}}|}{m} = \frac{mt\bar{\mathtt{D}}_{\mathrm{cr}}}{m} = t \frac{2}{\bar{\mathtt{D}}_{\mathrm{msc}}} \ln\Big(1 + \frac{\bar{\mathtt{D}}_{\mathrm{msc}}}{2}\Big). \tag{3.9}$$

It is easy to check that this value is always smaller than the average chain length t before the removal of chain merges. Even though we are keeping the longest chain from among any set of merging chains, the longer chains are more likely to merge into one another and be discarded.

The information which we have gathered so far can be also applied to analyze the parallel version of the perfect DP tradeoff, which will be presented in Chapter 4, because these two DP variants share their pre-computation phase and the information related to a perfect DP matrix does not depend on how pre-computation tables are treated.

Unlike other results of this work, our next claim is mostly based on experimental evidences, rather than on purely theoretical arguments. Note that the processing of a perfect DP table can bring about at most one alarm, which requires the partial regeneration of a single pre-computation chain. We will later show in Section 3.1.3 that, for a sufficiently wide range of $\bar{\mathtt{D}}_{\mathrm{msc}}$ of interest, the value computed through the formula

$$t \times \frac{1+0.577\,\bar{\mathtt{D}}_{\mathrm{msc}}}{1+0.451\,\bar{\mathtt{D}}_{\mathrm{msc}}} \tag{3.10}$$

agrees accurately with the experimentally obtained average number of one-way function iterations required for this single pre-computation chain regeneration to resolve an alarm.

Let us clarify that we are not claiming formula (3.10) to be *correct* in any theoretical sense. Our only claim here is that formula (3.10) predicts the average cost of resolving each alarm with accuracy that is more than sufficient for most practical purposes.

Proposition 3.4. *The online processing of a single perfect DP table is expected to require*

$$t \times \frac{1+0.577\,\bar{\mathtt{D}}_{\mathrm{msc}}}{1+0.451\,\bar{\mathtt{D}}_{\mathrm{msc}}}\,\frac{\bar{\mathtt{D}}_{\mathrm{msc}}}{1+\bar{\mathtt{D}}_{\mathrm{msc}}}. \tag{3.11}$$

invocations of the one-way function in relation to the resolving of a possible alarm.

Proof. Since amount of F-invocations to resolve an alarm is already obtained, we only need to compute the probability of encountering an alarm when working with a perfect DP table.

An online chain will merge into a perfect pre-computation matrix $\bar{\mathtt{DM}}$ if and only if it merges into the corresponding non-perfect pre-computation matrix $\mathtt{DM}$. Since we already know from (3.5) that the number of elements contained in $\mathtt{DM}$ as mt, the probability of encountering an alarm can be stated as

$$\sum_{i=0}^{\infty}\Big(1-\frac{1}{t}-\frac{mt}{\mathsf{N}}\Big)^{i}\frac{mt}{\mathsf{N}} = \frac{\frac{mt}{\mathsf{N}}}{\frac{1}{t}+\frac{mt}{\mathsf{N}}} = \frac{\bar{\mathtt{D}}_{\mathrm{msc}}}{1+\bar{\mathtt{D}}_{\mathrm{msc}}}.$$

The claimed expected cost of dealing with a possible alarm can be reached by multiplying this probability with the work factor (3.10). □

Having obtained the cost of dealing with alarms, the online complexities of the perfect DP tradeoff can be presented as a time memory tradeoff curve and its tradeoff coefficient is stated as below.

Theorem 3.1. *The time memory tradeoff curve for the perfect DP tradeoff is* $TM^2 = \bar{\mathsf{D}}_{TM^2}\mathsf{N}^2$*, where the tradeoff coefficient is given by*

$$\bar{\mathsf{D}}_{TM^2} = \Big(1 + \frac{1+0.577\,\bar{\mathsf{D}}_{\mathrm{msc}}}{1+0.451\,\bar{\mathsf{D}}_{\mathrm{msc}}}\,\frac{\bar{\mathsf{D}}_{\mathrm{msc}}}{1+\bar{\mathsf{D}}_{\mathrm{msc}}}\Big)\,\frac{\bar{\mathsf{D}}_{\mathrm{ps}}\big\{\ln(1-\bar{\mathsf{D}}_{\mathrm{ps}})\big\}^2}{\bar{\mathsf{D}}_{\mathrm{msc}}\,\bar{\mathsf{D}}_{\mathrm{cr}}^3}.$$

Proof. Since a single perfect DP matrix contains the correct answer to a given inversion problem with probability $\frac{mt\bar{\mathsf{D}}_{\mathrm{cr}}}{\mathsf{N}}$ by the definition of a coverage rate, the probability for the i-th DP table to be processed during the online phase executed for a single inversion target is $\big(1-\frac{mt\bar{\mathsf{D}}_{\mathrm{cr}}}{\mathsf{N}}\big)^{i-1}$, since each pre-computation table is processed one by one. The online processing of each table is expected to require t invocations of the one-way function for the online chain generation and the expected number of iterations required to resolve the alarm that could occur is given by Proposition 3.4. Hence, the number of one-way function iterations expected during the online phase of the perfect DP tradeoff is

$$\begin{aligned}
T &= \sum_{i=1}^{\ell}\Big(1-\frac{mt\bar{\mathsf{D}}_{\mathrm{cr}}}{\mathsf{N}}\Big)^{i-1}\Big(1+\frac{1+0.577\,\bar{\mathsf{D}}_{\mathrm{msc}}}{1+0.451\,\bar{\mathsf{D}}_{\mathrm{msc}}}\,\frac{\bar{\mathsf{D}}_{\mathrm{msc}}}{1+\bar{\mathsf{D}}_{\mathrm{msc}}}\Big)t\\
&= \frac{1-\big(1-\frac{mt\bar{\mathsf{D}}_{\mathrm{cr}}}{\mathsf{N}}\big)^{\ell}}{\frac{mt\bar{\mathsf{D}}_{\mathrm{cr}}}{\mathsf{N}}}\Big(1+\frac{1+0.577\,\bar{\mathsf{D}}_{\mathrm{msc}}}{1+0.451\,\bar{\mathsf{D}}_{\mathrm{msc}}}\,\frac{\bar{\mathsf{D}}_{\mathrm{msc}}}{1+\bar{\mathsf{D}}_{\mathrm{msc}}}\Big)t\\
&= \frac{\bar{\mathsf{D}}_{\mathrm{ps}}}{\bar{\mathsf{D}}_{\mathrm{msc}}\bar{\mathsf{D}}_{\mathrm{cr}}}\Big(1+\frac{1+0.577\,\bar{\mathsf{D}}_{\mathrm{msc}}}{1+0.451\,\bar{\mathsf{D}}_{\mathrm{msc}}}\,\frac{\bar{\mathsf{D}}_{\mathrm{msc}}}{1+\bar{\mathsf{D}}_{\mathrm{msc}}}\Big)t^2,
\end{aligned}$$

where the final equality relies on (3.7) or Proposition 3.2.

On the other hand, since each pre-computation table contains m entries and there are ℓ tables, the storage complexity of the perfect DP tradeoff is $M = m\ell$.

The time memory tradeoff curve for the perfect DP tradeoff is obtained by combining the complexities T and M as follows:

$$\begin{aligned}
TM^2 &= \frac{\bar{\mathsf{D}}_{\mathrm{ps}}}{\bar{\mathsf{D}}_{\mathrm{msc}}\bar{\mathsf{D}}_{\mathrm{cr}}}\Big(1+\frac{1+0.577\,\bar{\mathsf{D}}_{\mathrm{msc}}}{1+0.451\,\bar{\mathsf{D}}_{\mathrm{msc}}}\,\frac{\bar{\mathsf{D}}_{\mathrm{msc}}}{1+\bar{\mathsf{D}}_{\mathrm{msc}}}\Big)\,(mt\ell)^2 \\
&= \frac{\bar{\mathsf{D}}_{\mathrm{ps}}}{\bar{\mathsf{D}}_{\mathrm{msc}}\bar{\mathsf{D}}_{\mathrm{cr}}}\Big(1+\frac{1+0.577\,\bar{\mathsf{D}}_{\mathrm{msc}}}{1+0.451\,\bar{\mathsf{D}}_{\mathrm{msc}}}\,\frac{\bar{\mathsf{D}}_{\mathrm{msc}}}{1+\bar{\mathsf{D}}_{\mathrm{msc}}}\Big)\Big\{\frac{\ln(1-\bar{\mathsf{D}}_{\mathrm{ps}})}{\bar{\mathsf{D}}_{\mathrm{cr}}}\Big\}^2\mathsf{N}^2.
\end{aligned}$$

The second equality here is obtained through another application of (3.7). □

Let us clarify that both Proposition 3.4 and Theorem 3.1 depend on the empirical result (3.10). Both claims should be understood as providing practical formulas that can be used in practice to predict the behavior of the perfect DP tradeoff. They should not be taken as results that are *theoretically correct* in any sense.

3.1.2 Storage Optimization

An analysis of the perfect DP tradeoff would not be complete without a discussion of the storage optimization techniques.

Dealing with the storage size of the starting points is quite straightforward. One requires $\log\vec{m}_0$ bits of space for every starting point, and (3.3) implies that this will be one or two bits more than $\log m$ for parameters of interest. Hence, one may safely claim that the number of bits required to store a single starting point for a perfect DP tradeoff is very close to that required for the non-perfect DP tradeoff, when comparable parameters are used by the two algorithms.

As for record of an ending point, since every ending point is a DP, it suffices to consider truncations of just the DPs, rather than the general points of the search space $\mathcal{N}$. We will refer to the set of all possible truncated points as the *truncated space* and refer to the surjective map which sends each DP to its truncated form as the *truncation map*. A typical truncation map with a truncated space of size r simply retains $\log r$ bits of the ending point that are unrelated to the DP definition.

To effects of ending point truncation on the perfect DP tradeoff is slightly different from that on the non-perfect DP tradeoff, which was treated in [18]. The truncation may cause two non-merging pre-computation chains to become indistinguishable at the ending points and cause more chains to be discarded during the pre-computation phase. However, the following lemma shows that these further collisions can mostly be avoided by recording slightly more than $\log m$ bits.

Lemma 3.6. *Consider a truncation map with a truncated space of size r that is much smaller than the DP space. When $m = O(r)$ distinct ending point DPs are truncated, we can expect to obtain $r\{1-\exp(-\frac{m}{r})\}$ distinct truncated ending points. Conversely, when $r > m$, one must expect to truncate $r\ln\left(\frac{r}{r-m}\right)$ DPs in order to collect m distinct truncated points.*

This lemma is a trivial consequence of treating the truncation process as the random selection of points from a pool of r-many points and the aimed number of points as its image size. More precisely, the first claimed formula is trivially obtained in the same sense with the proofs of Lemma 3.1 and Lemma 3.3. The converse claim is an easy calculation of this.

Let us consider a specific example. When the truncated space is of size $r = 2^5 m$, it suffices to truncate $32m\ln\left(\frac{32}{31}\right) = 1.01596m$ DPs in order to obtain m distinct truncated ending points. Combining this information with Proposition 3.1, one can claim that, an increment of the required starting points by the ending point truncation, which leads to growth of the total pre-computation time, can be controlled within 3.1% at least, by recording just $5+\log m$ bits of each ending point. Note that this is not 1.596% and only claimed approximately, because the variable m appears not only in the $\frac{mt\ell}{\mathsf{N}}$ term of Proposition 3.1, but also inside the $\frac{\bar{\mathsf{D}}_{\mathrm{msc}}}{2}$ term. To be more precise, the rate of increase in the pre-computation cost, caused by retaining only $5+\log m$ bits of each ending point, can be stated as $\frac{(2+1.01596\bar{\mathsf{D}}_{\mathrm{msc}})\ 1.01596}{2+\bar{\mathsf{D}}_{\mathrm{msc}}}$, where $\bar{\mathsf{D}}_{\mathrm{msc}} = \frac{mt^2}{\mathsf{N}}$. In any case, the effects of ending point truncation on the collision of ending points can be maintained at an ignorable level by retaining a little more than $\log m$ bits of information through the truncation process. Note that by ignoring the ending point collisions induced by truncations, we are also ignoring their effects on the pre-computation time and also on the coverage rate, or, equivalently, the success probability.

We now need to discuss the effects of truncation on the online time. The terminating DP of the online chain must be searched for among the truncated ending points, so we have the possibility of truncation-related false alarm and then regenerating the pre-computation chain to resolve this alarm.

Lemma 3.7. *Consider a truncation map with a truncated space of size r. Assume that the truncated space is much smaller than the DP space and that r has been chosen to be large enough for the occurrences of indistinguishable ending points caused by truncations to be sufficiently limited. Then the number of extra one-way function invocations induced by truncation-related alarms is expected to be*

$$t\,\frac{m}{r}\,\frac{2}{\bar{\mathtt{D}}_{\mathrm{msc}}(1+\bar{\mathtt{D}}_{\mathrm{msc}})}\ln\Big(1+\frac{\bar{\mathtt{D}}_{\mathrm{msc}}}{2}\Big),$$

for each fully processed perfect DP table.

Proof. Let us compute the probability for an online chain to become a DP chain of length i and not merge into the perfect DP matrix, but have a truncated ending point that coincides with a truncated ending point in the perfect DP table. For this event to occur, the online chain must be created in the following manner: (1) The first i nodes of the online chain, starting from the correct pre-image of the inversion target, must be chosen among the non-DPs that do not belonging to the corresponding non-perfect DP matrix `DM`; (2) The final point must be chosen among DPs that are different from the m ending points in the perfect DP table; (3) Furthermore, the final point also must be chosen so that its truncated online ending point matches one of the m truncated ending points. The processes (2) and (3) are not quite independent, but since the number of DPs is much greater than the number of ending points, i.e., $\frac{\mathtt{N}}{t} \gg m$, the dependence can be ignored.

Thus, the probability we seek is

$$\Big(1-\frac{1}{t}-\frac{|\mathtt{DM}|}{\mathtt{N}}\Big)^i\Big(\frac{1}{t}-\frac{m}{\mathtt{N}}\Big)\frac{m}{r} \approx \Big(1-\frac{1}{t}-\frac{|\mathtt{DM}|}{\mathtt{N}}\Big)^i\frac{1}{t}\frac{m}{r} = \Big(1-\frac{1+\bar{\mathtt{D}}_{\mathrm{msc}}}{t}\Big)^i\frac{1}{t}\frac{m}{r},$$

where we have used $\frac{m}{\mathtt{N}} = O(\frac{1}{mt}) = o(\frac{1}{t})$ for the approximation and (3.5) for the final equality. Thus, the probability for the online processing of a perfect DP table to cause a truncation-related alarm is given by

$$\sum_{i=1}^{\infty}\Big(1-\frac{1+\bar{\mathtt{D}}_{\mathrm{msc}}}{t}\Big)^i\frac{1}{t}\frac{m}{r} = \frac{1-\frac{1+\bar{\mathtt{D}}_{\mathrm{msc}}}{t}}{\frac{1+\bar{\mathtt{D}}_{\mathrm{msc}}}{t}}\frac{1}{t}\frac{m}{r} \approx \frac{1}{1+\bar{\mathtt{D}}_{\mathrm{msc}}}\frac{m}{r}.$$

Notice that how likely a pre-computation chain is to be involved in a truncation-related alarm is independent of its length. Hence, the number of iterations required to regenerate the pre-computation chain involved with such a truncation-related alarm is expected to be the average chain length of the perfect DP matrix, which is given by (3.9). Thus, the cost of resolving alarms that are induced by truncation is

$$\frac{1}{1+\bar{\mathtt{D}}_{\mathrm{msc}}}\frac{m}{r}t\,\frac{2}{\bar{\mathtt{D}}_{\mathrm{msc}}}\ln\Big(1+\frac{\bar{\mathtt{D}}_{\mathrm{msc}}}{2}\Big),$$

for the full processing of a single perfect DP table. □

The normal one-way function iterations required to generate the online chain and deal with a possible alarm while processing a single perfect DP table was stated during the proof of Theorem 3.1 to be

$$\Big(1+\frac{1+0.577\,\bar{\mathtt{D}}_{\mathrm{msc}}}{1+0.451\,\bar{\mathtt{D}}_{\mathrm{msc}}}\,\frac{\bar{\mathtt{D}}_{\mathrm{msc}}}{1+\bar{\mathtt{D}}_{\mathrm{msc}}}\Big)t. \tag{3.12}$$

If we assume that sufficient information is left after the ending point truncation so that the number of indistinguishable ending points are kept small enough to be ignored, then, with parameters satisfying $\bar{\mathtt{D}}_{\mathrm{msc}} = 2$, the expected numbers of normal iterations and truncation-related iterations become $1.75499t$ and $\frac{2}{6}\ln(2)\frac{m}{r}t = 0.231049\frac{m}{r}t$, respectively. For example, with $r = 2^5m$, the ending point truncation increases the number of one-way function iterations by a mere $\frac{0.231049\frac{1}{32}t}{1.75499t} \approx 0.41\%$. The following can be stated for the general situation.

Proposition 3.5. *Suppose that the online phase of a perfect DP tradeoff implementation that stores each ending point in full requires T iterations of the one-way function to complete. Consider a truncation map for which the truncated space is of size $r = 2^{\varepsilon}m$. If ε is large enough for the occurrences of indistinguishable ending points caused by truncations to be ignored, then the implementation with the ending point truncation requires*

$$\frac{2\ln\left(1+\frac{\bar{\mathtt{D}}_{\mathrm{msc}}}{2}\right)}{\bar{\mathtt{D}}_{\mathrm{msc}}(1+\bar{\mathtt{D}}_{\mathrm{msc}})\left(1+\frac{1+0.577\bar{\mathtt{D}}_{\mathrm{msc}}}{1+0.451\bar{\mathtt{D}}_{\mathrm{msc}}}\frac{\bar{\mathtt{D}}_{\mathrm{msc}}}{1+\bar{\mathtt{D}}_{\mathrm{msc}}}\right)}\frac{T}{2^{\varepsilon}}$$

additional iterations of the one-way function to complete.

For parameters satisfying $\bar{\mathtt{D}}_{\mathrm{msc}} = 2$, the above is $\frac{0.231049}{1.75499}\frac{T}{2^{\varepsilon}} = 0.131652\frac{T}{2^{\varepsilon}}$. This implies that, for parameters of interest, a small ε is enough to keep the negative effects of ending point truncation on the online time to a reasonably small level.

Let us summarize the situation concerning the storage of each perfect DP table entry. The starting point can be stored using slightly more than $\log m$ bits. Ending point DPs can be truncated so that a little more than $\log m$ bits of information is retained with very little negative effect on the success probability, pre-computation cost, and online time. The index file technique can be used to remove almost $\log m$ further bits per ending point without any loss of information. In conclusion, storage of each starting point and ending point pair requires a little more than $\log m$ bits. Therefore, the perfect DP tradeoff requires the same amount of bits to record each table entry as those for the original and parallel DP tradeoffs.

3.1.3 Experiment Results

We have verified the correctness of major parts of our complexity analysis with experiments. For the first two sets of our experiments, the one-way function was

Table 3.1: The number of DP chains before and after removal of chain merges and the coverage rate of the perfect DP matrix. ($\mathsf{N} = 2^{40}$; $\hat{t} = 15t$).

m	t	$\bar{\mathsf{D}}_{\mathrm{msc}}$	$\vec{m}_0$ used	test m	theoretical $\bar{\mathsf{D}}_{\mathrm{cr}}$	test $\bar{\mathsf{D}}_{\mathrm{cr}}$
2000	2^{14}	0.48828	2488	2000.88	0.89475	0.89302
4000	2^{14}	0.97656	5953	3996.01	0.81433	0.81412
6000	2^{14}	1.46484	10394	5996.79	0.75028	0.74934
10000	2^{13}	0.61035	13051	10005.45	0.87274	0.87319
20000	2^{13}	1.22070	32207	20001.52	0.78062	0.78079
30000	2^{13}	1.83105	57465	30003.72	0.70997	0.71020

instantiated with the key to ciphertext mapping, under a randomly fixed plaintext, of the block cipher AES-128. Freshly generated random plaintexts were used to create different one-way functions that were required for repetitions of the same test. Bit-masking of ciphertexts to 40 bits and its zero-extension to 128-bit keys were used to restrict the search space to a manageable size of $\mathsf{N} = 2^{40}$.

The first experiment was designed to verify Lemma 3.2 and Proposition 3.3 simultaneously. Recall that Lemma 3.2 related the number of starting points to the number of distinct ending points in a non-perfect DP matrix and that Proposition 3.3 presented the coverage rate of the perfect DP matrix.

After fixing suitable parameters m and t, we first computed the $\vec{m}_0$ value, as specified by (3.3). We generated chains from $\vec{m}_0$ distinct starting points and recorded their terminating DPs, together with their respective chain lengths. A small number of chains that extended beyond the moderately large chain length bound of $\hat{t} = 15t$ were discarded during this process. After dealing with chain merges by retaining only the information corresponding to the longest chain among any set of merging chains, the number of remaining DPs was counted. Next, the lengths of the surviving chains were added together and taken as the number of distinct entries in the perfect DP matrix. The obtained count of matrix entries, divided by mt, is our test $\bar{\mathsf{D}}_{\mathrm{cr}}$ value. The whole process was repeated 200 times for each choice of parameter set and the obtained values were averaged.

The test results are summarized in Table 3.1, together with the integer $\vec{m}_0$

values we have used and the theoretically computed coverage rates. In each row, the reported number of distinct ending points that resulted from our theoretically computed $\vec{m}_0$ starting points is very close to the targeted m value, in spite of the small number of test repetitions. It can also be seen that our theory was able to predict the coverage rates accurately.

Even though this test gives some confidence as to the correctness of our theory, let us present another test that makes sure that our accurate predictions of the coverage rate did not result from some lucky averaging effect that conveniently hid logical errors in our lower level arguments.

Recall that the proof of Proposition 3.3 relied heavily on our ability to write the probability for a random chain of length k not to merge into any of the chains in a non-perfect DP matrix that are longer than k. More specifically, this probability was taken to be

$$\left\{1+\frac{\bar{\mathtt{D}}_{\mathrm{msc}}}{2}\left(1-e^{-\frac{k}{t}}\right)\right\}^{-2} \tag{3.13}$$

and was interpreted as the probability for a chain in a non-perfect DP matrix to survive through the process of removing chain merges.

To test this core logic, we first generated multiple non-perfect DP matrices, discarding the small number of chains reaching the length bound of $\hat{t} = 15t$. Then, for each $1 \leq k < \hat{t}$, we counted and recorded the total number of chains of length k found among these matrices. Next, we removed merges from each of the DP matrices to create multiple perfect DP matrices and, once again, recorded the number of chains of each length. We took the ratio of the two chain counts, for each length k, as our test value of the probability for chains of length k to survive through the chain merge removal process. Note that this ratio of counts cannot be computed separately for each DP matrix and then later averaged over multiple DP matrices, since the number of chains of any given length is likely to be very small and often zero for any single DP matrix.

The test results are provided by Figure 3.1. The probability (y-axis) for chain survival through the chain merge removal process is given for each chain length

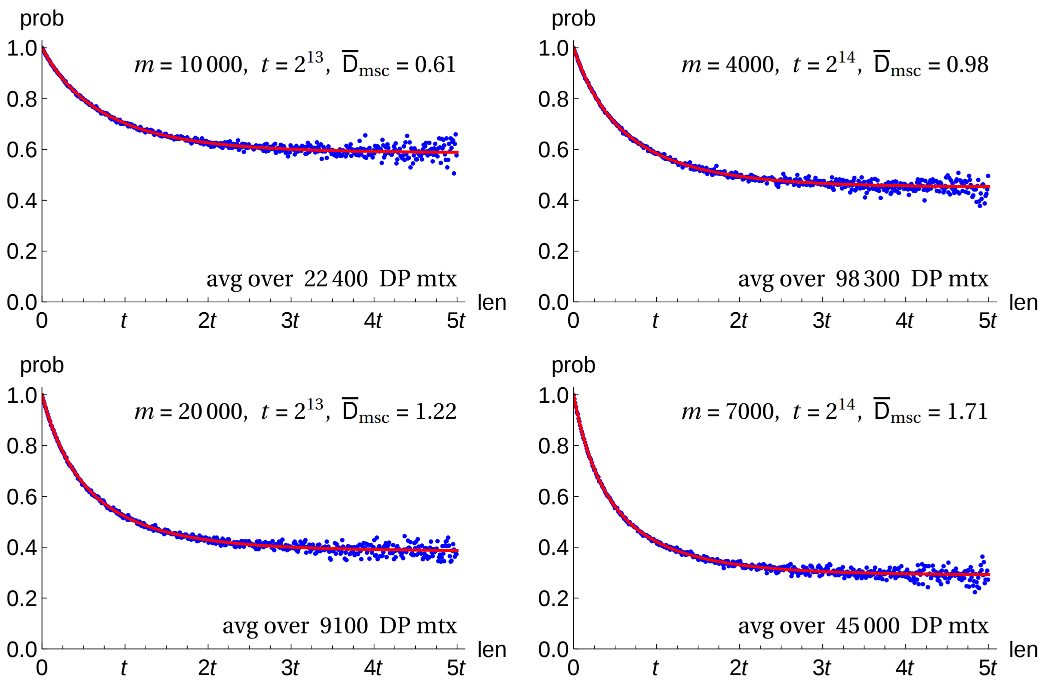

Figure 3.1: The probability for DP chains of each length to survive through the treatment of merging chains in a non-perfect DP matrix. (test: *dots*; theory: *line*; $\mathsf{N} = 2^{40}$; $\hat{t} = 15t$).

(x-axis). The lines correspond to our theory, as given by (3.13), and the dots represent the count ratios obtained through tests. Even though our chain length bound was $\hat{t} = 15t$, we have displayed the data only for chain lengths less than approximately $5t$. Furthermore, in each box, we only plotted approximately 500 dots that are equally spaced in terms of chain length values, since densely packing all $5t$ dots into each box made the graphs harder to comprehend.

The experimental data agrees well with our theory in all the boxes. Notice that the test results are less reliable at the large chain lengths. This is because longer DP chains appear less frequently and these large chain length data were obtained from a smaller number of chains. A much larger number of DP matrices would need to be generated to obtain meaningful test values at lengths much larger than $5t$.

Our final experiment measured the cost of regenerating the pre-computation chain for each online chain that produces an alarm. For this purpose, a slightly

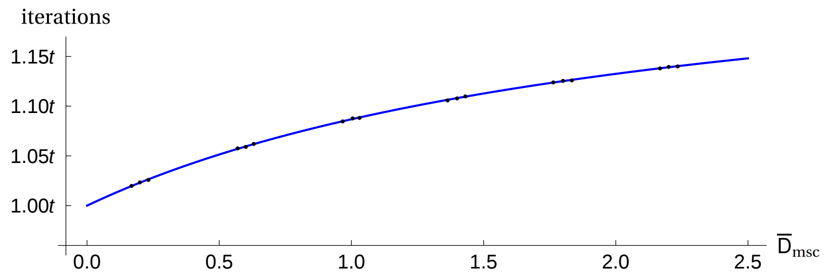

Figure 3.2: The number of one-way function iterations required to resolve each alarm for the perfect DP tradeoff, plotted in relation to the $\bar{\mathsf{D}}_{\text{msc}}$ value for the parameters that were used. (test: *dots*; theory: *line*; $\mathsf{N} = 2^{48}$; $\hat{t} = 15t$).

modified version of the MD5 hash function that accepts inputs of fixed 48-bit length was used as the one-way function. Recall that MD5 operates iteratively on 512-bit segments of its input. Since the length of our inputs was fixed, rather than conforming precisely to the length-related padding scheme specified for MD5, we placed the 48-bit input at the least significant end of a 512-bit block and filled the remaining 464 bits with zeros, before applying the usual 4-round/64-step operations of the MD5. Likewise, the least significant 48 bits of the 128-bit MD5 output were taken as the output of our one-way function. Note that this modified version of the MD5 function will be also used for all experiments in Section 4.3

For each choice of $\vec{m}_0$ and t, we created multiple perfect DP tables from $\vec{m}_0$ starting points. For each pre-computation table, we generated as many online chains as was required to observe a sufficiently large number of alarms. For each merge, the associated pre-computation chain was generated, up to the point of merge, and the length of this chain segment was recorded. That is, the online chain record technique, previously explained in Section 2.1.1, was used to terminate the chain regeneration at the point of chain merge, rather than at the ending point DP.

The results of our experiments, together with the predictions given by formula (3.10), are summarized in Table 3.2. We have also plotted the experiment data of Table 3.2 and the curve given by formula (3.10) in Figure 3.2. The test

Table 3.2: The number of one-way function iterations required to resolve each alarm for various parameters. ($\mathsf{N} = 2^{48}$; $\hat{t} = 15t$).

parameters						test	formula	$\frac{\text{test}}{\text{formula}}$
$\vec{m}_0$	m	t	#(tbl)	#(alarm) / tbl	$\bar{\mathsf{D}}_{\mathrm{msc}}$	$\frac{1}{t} \times$ #(itr)	$\frac{1}{t} \times$ Eq.(3.10)	
3000	2766	131072	1280	10000	0.16882	1.02002	1.01977	1.00025
231000	209976	16384	128	30000	0.20025	1.02354	1.02314	1.00039
17000	15230	65536	640	10000	0.23239	1.02616	1.02650	0.99967
48000	37354	65536	128	10000	0.56998	1.05758	1.05713	1.00042
12775	9843	131072	640	5000	0.60077	1.05889	1.05956	0.99937
870000	661405	16384	128	20000	0.63077	1.06202	1.06187	1.00014
1504000	1013856	16384	128	20000	0.96689	1.08460	1.08483	0.99978
99000	65884	65536	128	10000	1.00531	1.08758	1.08715	1.00039
6400	4223	262144	1280	10000	1.03101	1.08824	1.08867	0.99961
9400	5588	262144	1280	10000	1.36426	1.10606	1.10642	0.99967
156000	91761	65536	128	20000	1.40016	1.10806	1.10814	0.99993
2576000	1501280	16384	128	20000	1.43173	1.10999	1.10962	1.00033
217500	115580	65536	128	20000	1.76361	1.12370	1.12377	0.99994
3587000	1887750	16384	128	20000	1.80030	1.12553	1.12519	1.00030
14400	7512	262144	1280	10000	1.83398	1.12610	1.12647	0.99967
74000	35512	131072	128	10000	2.16748	1.13818	1.13810	1.00007
4845000	2307050	16384	128	20000	2.20017	1.13976	1.13915	1.00054
310000	146425	65536	128	20000	2.23427	1.14000	1.14022	0.99981

value given in each row of the table is an average obtained after creating "#(tbl)"-many tables and generating, for each table, as many online chains as were required to obtain "#(alarm)/tbl"-many alarms. Each value computed through formula (3.10) is very close to the average number of one-way function iterations required per alarm that was obtained experimentally. Also, after viewing Figure 3.2, one can be confident that formula (3.10) will be quite accurate, at least for all parameter choices satisfying $0 < \bar{\mathsf{D}}_{\mathrm{msc}} < 2.3$.

3.2 Analysis of the Perfect Rainbow Tradeoff

In this section, we present information of the perfect rainbow tradeoff required for comparison of tradeoff algorithms. Even though much of the material given here have not appeared before in the form presented here, the technical core of our complexity analyses were developed by previous works, and the arguments and proofs of this section contain no new ideas. These certainly require some work to obtain, but, given enough time, anyone with a full understanding of the papers [24], [15], and [18] should be able to reproduce the claims of this section.

3.2.1 Online Efficiency

Unlike the perfect DP tradeoff case, the difficult parts of the complexity analysis for the perfect rainbow tradeoff have already been done by previous works, and it only remains to combine these.

Lemma 3.8. *A non-perfect rainbow matrix created with m_0 starting points is expected to contain $\frac{2m_0}{2+\mathsf{pR}_{\mathrm{msc}}}$ distinct ending points, where $\mathsf{pR}_{\mathrm{msc}} = \frac{m_0 t}{\mathsf{N}}$. Conversely, given m, one must generate $m_0 = \frac{2}{2-\bar{\mathsf{pR}}_{\mathrm{msc}}} m$ chains, where $\bar{\mathsf{pR}}_{\mathrm{msc}} = \frac{mt}{\mathsf{N}}$, in order for m to be the expected number of chains contained in the corresponding perfect rainbow matrix.*

Proof. Consider a non-perfect rainbow matrix created with m_0 starting points. It is known [6, 15] that the number of distinct points m_i expected in the i-th column of this matrix, satisfies

$$\frac{m_i}{\mathsf{N}} = \frac{1}{\frac{\mathsf{N}}{m_0} + \frac{i}{2}},$$

for each $0 \le i \le t$. Setting $i = t$ gives the number of distinct ending points

$$m_t = \frac{\mathsf{N}}{\frac{\mathsf{N}}{m_0} + \frac{t}{2}} = \frac{2m_0}{2 + \frac{m_0 t}{\mathsf{N}}},$$

which is the first claim of this lemma.

To obtain the second claim, it suffices to solve for m_0 from the relation

$$m = m_t = \frac{2m_0}{2 + \frac{m_0 t}{\mathsf{N}}}.$$

This is equivalent to

$$m_0 = \frac{2m}{2 - \frac{mt}{\mathsf{N}}},$$

which is the second claim. □

Throughout this thesis,

$$m_0 = \frac{2}{2 - \bar{\mathrm{pR}}_{\mathrm{msc}}}\, m \tag{3.14}$$

will always denote the number of starting points that are required to create a perfect rainbow table that is expected to contain m ending points. This is the value of m_0 that should be used by Algorithm 9, given the algorithm parameters m and t.

Let us discuss about an interesting situation, which we will refer to as the maximal perfect rainbow tradeoff, is when $m_0 = \mathsf{N}$. Since a larger number of starting points bring about a larger number distinct ending points, this is when a perfect rainbow table is of maximum size [6, 24], assuming a fixed t. Substituting $m_0 = \mathsf{N}$ into the second equation in the proof of Lemma 3.8, we see that $m = m_t = \frac{2\mathsf{N}}{2+t}$.

This implies an upper bound

$$\bar{\mathsf{pR}}_{\mathrm{msc}} \leq \frac{2t}{t+2} < 2 \tag{3.15}$$

on the possible range of $\bar{\mathsf{pR}}_{\mathrm{msc}}$, with the possibility of $\bar{\mathsf{pR}}_{\mathrm{msc}}$ reaching very close to 2, for any practical t.

The pre-computation phase of a perfect rainbow tradeoff requires $m_0 t\ell$ iterations of the one-way function. As with the DP case, we define the *pre-computation coefficient* for the perfect rainbow tradeoff to be $\bar{\mathsf{pR}}_{\mathrm{pc}} = \frac{m_0 t\ell}{\mathsf{N}}$, so that the number of one-way function iterations required for the pre-computation phase becomes $\bar{\mathsf{pR}}_{\mathrm{pc}}\mathsf{N}$. The following statement is a direct consequence of Lemma 3.8.

Proposition 3.6. *The pre-computation coefficient of the perfect rainbow tradeoff is*

$$\bar{\mathsf{pR}}_{\mathrm{pc}} = \frac{2}{2-\bar{\mathsf{pR}}_{\mathrm{msc}}}\frac{mt\ell}{\mathsf{N}} = \frac{2\,\bar{\mathsf{pR}}_{\mathrm{msc}}}{2-\bar{\mathsf{pR}}_{\mathrm{msc}}}\ell.$$

The success probability of a perfect rainbow tradeoff may easily be stated [6] as

$$\bar{\mathsf{pR}}_{\mathrm{ps}} = 1-\Big(1-\frac{m}{\mathsf{N}}\Big)^{t\ell} = 1-\exp\Big(-\frac{mt\ell}{\mathsf{N}}\Big) = 1-\exp\big(-\bar{\mathsf{pR}}_{\mathrm{msc}}\ell\big), \tag{3.16}$$

and this shows that the choice of ℓ determines the matrix stopping constant

$$\bar{\mathsf{pR}}_{\mathrm{msc}} = -\frac{\ln(1-\bar{\mathsf{pR}}_{\mathrm{ps}})}{\ell}. \tag{3.17}$$

One must adhere to, when selecting parameters that achieve a prescribed probability of success. However, according to (3.15), the number of tables must satisfy

$$\ell > -\frac{1}{2}\ln(1-\bar{\mathsf{pR}}_{\mathrm{ps}}). \tag{3.18}$$

That is, to achieve a given success probability $\bar{\mathsf{pR}}_{\mathrm{ps}}$, the number of tables one must use is lower bound by (3.18). No set of parameters that uses a smaller number of tables can achieve the desired success probability.

Using Proposition 3.6, we can restate the probability of success (3.16) as follows.

Proposition 3.7. *The success probability of the perfect rainbow tradeoff is*

$$\bar{\mathrm{pR}}_{\mathrm{ps}} = 1 - \exp\Big(- \frac{2 - \bar{\mathrm{pR}}_{\mathrm{msc}}}{2}\, \bar{\mathrm{pR}}_{\mathrm{pc}} \Big).$$

The time memory tradeoff curve of the perfect rainbow tradeoff is obtained straightforward from existing works.

Theorem 3.2. *The time memory tradeoff curve for the perfect rainbow tradeoff is* $TM^2 = \bar{\mathrm{pR}}_{TM^2}\mathsf{N}^2$*, where the tradeoff coefficient is*

$$\begin{aligned}\bar{\mathrm{pR}}_{TM^2} = {} & \Big(\bar{\mathrm{pR}}_{\mathrm{msc}}\ell - \frac{\bar{\mathrm{pR}}_{\mathrm{msc}}}{2} + \ell - 2 + \frac{3}{2\ell}\Big) \\ & - \Big(\frac{\bar{\mathrm{pR}}_{\mathrm{msc}}^2 \ell}{4} + \bar{\mathrm{pR}}_{\mathrm{msc}}\ell^2 - \bar{\mathrm{pR}}_{\mathrm{msc}}\ell + \bar{\mathrm{pR}}_{\mathrm{msc}} + \ell - 2 + \frac{3}{2\ell}\Big) e^{-\bar{\mathrm{pR}}_{\mathrm{msc}}\ell}.\end{aligned}$$

Proof. According to [15], the expected number of one-way function iterations required to generate the online chain is

$$\ell \Big\{ 1 - (1 + \bar{\mathrm{pR}}_{\mathrm{msc}}\ell) e^{-\bar{\mathrm{pR}}_{\mathrm{msc}}\ell} \Big\} \Big(\frac{t}{\bar{\mathrm{pR}}_{\mathrm{msc}}\ell} \Big)^2,$$

and that required to resolve alarms is[1]

$$\Big(\Big\{ \bar{\mathrm{pR}}_{\mathrm{msc}}\Big(\ell - \frac{1}{2}\Big) - \Big(2 - \frac{3}{2\ell}\Big) \Big\} + \Big\{ \Big(2 - \frac{3}{2\ell}\Big) + \bar{\mathrm{pR}}_{\mathrm{msc}}(\ell - 1) - \frac{\bar{\mathrm{pR}}_{\mathrm{msc}}^2\ell}{4} \Big\} e^{-\bar{\mathrm{pR}}_{\mathrm{msc}}\ell} \Big) \Big(\frac{t}{\bar{\mathrm{pR}}_{\mathrm{msc}}\ell} \Big)^2.$$

The time complexity T is the sum of these two terms.

As with the DP tradeoff, the storage complexity associated with ℓ tables, each containing m entries is $M = m\ell$. The complexities T and M can be combined and easily simplified to arrive at the claimed statement. □

[1]The single $e^{c_{\mathrm{R}}}$ appearing in [15, p.312] should be corrected to $e^{c_{\mathrm{R}}\ell}$.

3.2.2 Storage Optimization

As was with the perfect DP tradeoff in Section 3.1.2, a single starting point for the perfect rainbow tradeoff can be recorded in $\log m_0$ bits, and (3.14) shows how this compares with $\log m$. However, unlike the DP case, since $\bar{\mathrm{pR}}_{\mathrm{msc}}$ may take values that are very close to 2, there is the possibility of $\log m_0$ being much larger than $\log m$.

A hint for resolving this problem comes from the derivation process of (3.15), which shows that $\bar{\mathrm{pR}}_{\mathrm{msc}}$ being close to 2 is associated with an unrealistically large amount of pre-computation as much as the dictionary attack. In any real-world situation, there will be a bound on the pre-computation cost one is willing to invest. So, let us combine Proposition 3.6 and (3.17), and consider a somewhat arbitrary bound of

$$\bar{\mathrm{pR}}_{\mathrm{pc}} = \frac{2}{2-\bar{\mathrm{pR}}_{\mathrm{msc}}}\big\{-\ln(1-\bar{\mathrm{pR}}_{\mathrm{ps}})\big\} \leq 20, \tag{3.19}$$

on the pre-computation coefficient. Unless the requirement on the success rate is unrealistically small, this gives a reasonably small bound on the coefficient $\frac{2}{2-\bar{\mathrm{pR}}_{\mathrm{msc}}}$ of (3.14), so that $\log m$ will be similar to $\log m_0$. This shows that, for any practical situation, it suffices to allocate slightly more than $\log m$ bits of storage to each starting point.

One side effect of (3.19) is that it implies the bound $\bar{\mathrm{pR}}_{\mathrm{ps}} \leq 1 - \frac{1}{e^{20}}$ on the success probability one can consider. However, a success probability that is arbitrarily close to 1 cannot be achieved without enormous amount of pre-computation. Thus, since $99.999999\% < 1 - \frac{1}{e^{20}}$, the above bound on the success probability will not make a problem to execute the perfect rainbow tradeoff and also the another implied bound on the success probability is essentially meaningless for even a moderately large bound on the pre-computation cost.

The ending point truncation technique is the subject of our next discussion. Unlike the case of the perfect DP tradeoff which truncated only for the DPs, ending points may take any form with the rainbow tradeoff, so we now consider the truncation of any point from $\mathcal{N}$. We will reuse the terms *truncation map* and *truncated space* that were previously introduced in Section 3.1.2. As was with the DP case, truncation may cause two ending points of a perfect rainbow table to become indistinguishable, and the following lemma, in a similar sense with Lemma 3.6, solves this problem.

Lemma 3.9. *Consider a truncation map with a truncated space of size $r \ll \mathsf{N}$. When $m = O(r)$ distinct ending points are truncated, we can expect to obtain $r\big\{1 - \exp(-\frac{m}{r})\big\}$ distinct truncated points. Conversely, given $r > m$, one must truncate $r \ln\big(\frac{r}{r-m}\big)$ distinct ending points in order for m to be the expected number of distinct truncated points.*

The example values that were given below Lemma 3.6 are still valid for the perfect rainbow tradeoff. That is, truncation of $1.01596m$ ending points will give m distinct truncated points, when $r = 2^5 m$. Hence, the effects of ending point truncation on pre-computation cost and success probability can be suppressed to an ignorable degree by the use of an r such that $\log r = \varepsilon + \log m$ for some small positive integer ε.

Analogous to the DP case, if required, one can work with (3.14) to find the correct m_0 value one must use in order to collect the slightly larger number of pre-computation chains that do not merge into each other. Note that our previous discussion of how $\bar{\mathsf{pR}}_{\mathrm{msc}}$ is sufficiently bounded away from 2, in practice, implies that the non-linearity hidden within $\bar{\mathsf{pR}}_{\mathrm{msc}}$ will not cause too much disturbance. In particular, our claim of each starting point requiring $\log m_0 \approx \log m$ bits of storage remains valid even if one wants to account for the small loss of pre-computation chains experienced through the truncation of ending points.

The effect of truncation on the online time is considered next.

Lemma 3.10. *Consider a truncation map with a truncated space of size r. Assume that $r \ll \mathsf{N}$ and that r has been chosen to be large enough for the occurrences of* *indistinguishable ending points caused by truncations to be sufficiently limited. Then, during the online phase of the perfect rainbow tradeoff, one can expected to observe*

$$\left\{\begin{array}{l}\Big(\bar{\mathrm{pR}}_{\mathrm{msc}}\ell^2-\bar{\mathrm{pR}}_{\mathrm{msc}}\ell+\frac{\bar{\mathrm{pR}}_{\mathrm{msc}}}{2}-\ell+2-\frac{3}{2\ell}\Big)\\ +\Big(\frac{\bar{\mathrm{pR}}_{\mathrm{msc}}^2\ell}{4}-\bar{\mathrm{pR}}_{\mathrm{msc}}\ell+\bar{\mathrm{pR}}_{\mathrm{msc}}+\ell-2+\frac{3}{2\ell}\Big)e^{-\bar{\mathrm{pR}}_{\mathrm{msc}}\ell}\end{array}\right\}\frac{m}{r}\Big(\frac{t}{\bar{\mathrm{pR}}_{\mathrm{msc}}\ell}\Big)^2$$

extra one-way function invocations induced by truncation-related alarms.

Proof. Consider the non-perfect rainbow matrix created with $m_0=\frac{2m}{2-\bar{\mathrm{pR}}_{\mathrm{msc}}}$ starting points and let m_i $(0\le i\le t)$ denote the number of distinct points expected in the i-th column of this matrix as before. Next, consider the online chain created at the i-th online iteration for the corresponding pre-computation table, i.e., the online chain of length i, starting from the correct inversion target pre-image. Note that this online chain will merge into the perfect rainbow matrix if and only if it merges into the non-perfect rainbow matrix. Treating the online chain as a random walk, the probability for this chain not to merge into the perfect rainbow matrix can be written as $\prod_{j=0}^{i}\big(1-\frac{m_{t-j}}{\mathsf{N}}\big)$.

On the other hand, assuming that r is large enough for the m truncated ending points to be distinct, the probability for the truncation of the ending point for the online chain that did not merge into the perfect matrix to colide one of the m truncated ending points is $\frac{m}{r}$. Hence, the probability for an online chain of length i to bring about a truncation-related false alarm is

$$\frac{m}{r}\prod_{j=0}^{i}\Big(1-\frac{m_{t-j}}{\mathsf{N}}\Big)=\frac{m}{r}\,\frac{\frac{\mathsf{N}}{m_0}+\frac{t-i-1}{2}}{\frac{\mathsf{N}}{m_0}+\frac{t}{2}}\,\frac{\frac{\mathsf{N}}{m_0}+\frac{t-i-2}{2}}{\frac{\mathsf{N}}{m_0}+\frac{t-1}{2}}$$

Here, one must substitute $\frac{m_i}{\mathsf{N}}=\big(\frac{\mathsf{N}}{m_0}+\frac{i}{2}\big)^{-1}$, which may be found in [6, 15], to obtain the equality. This may be approximated and further simplified to

$$\frac{m}{r}\Bigg(\frac{\frac{\mathsf{N}}{m_0}+\frac{t-i}{2}}{\frac{\mathsf{N}}{m_0}+\frac{t}{2}}\Bigg)^2=\frac{m}{r}\Bigg(1-\frac{\frac{i}{2}}{\frac{\mathsf{N}}{m_0}+\frac{t}{2}}\Bigg)^2=\frac{m}{r}\Bigg(1-\frac{1}{\frac{2\mathsf{N}}{m_0 t}+1}\frac{i}{t}\Bigg)^2=\frac{m}{r}\Bigg(1-\frac{\bar{\mathrm{pR}}_{\mathrm{msc}}}{2}\frac{i}{t}\Bigg)^2,$$

where the final equality results from the substitution of m_0, as given by (3.14).

Now, i-th online iterations for all ℓ pre-compuataion tables are processed i.e. the ℓ online chains of length i are generated if and only if all ℓ chains of length strictly smaller than i did not return the correct answer to the inversion problem, and this happens with probability $\big(1-\frac{m}{\mathsf{N}}\big)^{\ell(i-1)}$. Since each alarm from and online chain of length i requires $(t-i)$ iterations of the one-way function from the corresponding starting point to resolve, the expected number of extra one-way function iterations induced by truncation-related alarms can be written as

$$\ell\sum_{i=0}^{t}(t-i)\frac{m}{r}\Big(1-\frac{\bar{\mathrm{pR}}_{\mathrm{msc}}}{2}\frac{i}{t}\Big)^2\Big(1-\frac{m}{\mathsf{N}}\Big)^{\ell(i-1)}.$$

Rewriting this in the form

$$t^2\ell\frac{m}{r}\sum_{i=0}^{t}\Big(1-\frac{i}{t}\Big)\Big(1-\frac{\bar{\mathrm{pR}}_{\mathrm{msc}}}{2}\frac{i}{t}\Big)^2\exp\Big(-\frac{mt\ell}{\mathsf{N}}\frac{i-1}{t}\Big)\frac{1}{t},$$

we can see that, unless t is very small, the expected extra cost can be approximated by the definite integral

$$t^2\ell\frac{m}{r}\int_0^1(1-u)\Big(1-\frac{\bar{\mathrm{pR}}_{\mathrm{msc}}}{2}u\Big)^2\exp\big(-\bar{\mathrm{pR}}_{\mathrm{msc}}\ell u\big)\,du.$$

Claimed formula can be obtained by explicit computation of this definite integral. □

Recall that the total online time, without ending point truncation, was given during the proof of Theorem 3.2. Comparing the complexity with what is given by Lemma 3.10, it is straightforward to express the effects of ending point truncation on the total online time.

Proposition 3.8. *Suppose that the online phase of a perfect rainbow tradeoff implementation that stores each ending point in full requires T iterations of the one-way function to complete. Consider a truncation map for which the truncated space is of size $r = 2^{\varepsilon} m \ll \mathsf{N}$. If ε is large enough for the occurrences of indistinguishable ending points caused by truncations to be ignored, then the implementation with the ending point truncation requires*

$$\frac{-\left(\frac{3}{2\ell} - 2 + \ell - \bar{\mathrm{pR}}_{\mathrm{msc}}\left(\frac{1}{2} - \ell + \ell^2\right)\right) + \left(\frac{3}{2\ell} - 2 + \ell + \bar{\mathrm{pR}}_{\mathrm{msc}}\left(1 - \ell + \frac{\bar{\mathrm{pR}}_{\mathrm{msc}}\ell}{4}\right)\right)e^{-\bar{\mathrm{pR}}_{\mathrm{msc}}\ell}}{\left(\frac{3}{2\ell} - 2 + \ell - \bar{\mathrm{pR}}_{\mathrm{msc}}\left(\frac{1}{2} + \ell\right)\right) - \left(\frac{3}{2\ell} - 2 + \ell + \bar{\mathrm{pR}}_{\mathrm{msc}}\left(1 - \ell + \ell^2 + \frac{\bar{\mathrm{pR}}_{\mathrm{msc}}\ell}{4}\right)\right)e^{-\bar{\mathrm{pR}}_{\mathrm{msc}}\ell}} \frac{T}{2^{\varepsilon}}$$

additional iterations of the one-way function to complete.

For parameters satisfying $\bar{\mathrm{pR}}_{\mathrm{msc}} = 1$ and $\ell = 1$, the claim is that $0.774568\frac{T}{2^{\varepsilon}}$ additional one-way function iterations are expected due to truncation-related alarms. At $\varepsilon = 5$, this is $0.0242052T$, which implies a 2.42% increase in online time due to ending point truncation.

Let us gather the information about the storage optimization for the perfect rainbow tradeoff. Each starting point can be recorded in slightly more than $\log m$ bits. Each ending point can be truncated to slightly more than $\log m$ bits with little effect on the success probability, pre-computation cost, and online time. The index file technique can be used to remove almost $\log m$ further bits per ending point without any loss of information. In all, storage of each starting point and ending point pair requires a little more than $\log m$ bits. Even though this is identical to the conclusion obtained in [18] for the non-perfect rainbow tradeoff, those inner workings are different so that the analysis had to be repeated here for the perfect rainbow tradeoff.

Chapter 4

Perfect Parallel DP Tradeoff

In this chapter, a full analysis of the perfect parallel DP $\bar{\mathtt{pD}}$ tradeoff, which is a mainly interesting tradeoff algorithm in this work and was described in Section 2.1.1, will be provided. As mentioned before, in a recent talk [26], it is announced that the $\bar{\mathtt{pD}}$ tradeoff outperforms the perfect rainbow tradeoff based on experimental results. The claim can be verified, utilizing the information gathered in this section.

4.1 Online Efficiency

The goal of this section is to obtain the computational complexity of the $\bar{\mathtt{pD}}$ tradeoff. Note that the only difference with the perfect DP tradeoff which analysis was presented in Section 3.1, is in the order of operations executed during their online phases. Hence, many formulas known for the perfect DP tradeoff hold true for the $\bar{\mathtt{pD}}$ tradeoff, when $\bar{\mathtt{D}}_{\mathrm{msc}}$ and $\bar{\mathtt{D}}_{\mathrm{ps}}$ are replaced by $\bar{\mathtt{pD}}_{\mathrm{msc}}$ and $\bar{\mathtt{pD}}_{\mathrm{ps}}$, respectively.

Note that one can rewrite the pre-computation coefficient $\bar{\mathtt{D}}_{\mathrm{pc}}$ as the following formula

$$\bar{\mathtt{D}}_{\mathrm{pc}} = -\frac{\bar{\mathtt{pD}}_{\mathrm{msc}}}{2}\Big(1+\frac{\bar{\mathtt{pD}}_{\mathrm{msc}}}{2}\Big)\frac{\ln(1-\bar{\mathtt{pD}}_{\mathrm{ps}})}{\ln(1+\frac{\bar{\mathtt{pD}}_{\mathrm{msc}}}{2})}, \tag{4.1}$$

by combining Proposition 3.1, (3.7), and Proposition 3.3. Because of its frequent appearances within this section, we will often use the shorthand notation

$$\mathtt{p\bar{D}}_{\mathrm{ln}} = -\frac{\ln(1-\mathtt{p\bar{D}}_{\mathrm{ps}})}{\ln(1+\frac{\mathtt{p\bar{D}}_{\mathrm{msc}}}{2})}. \tag{4.2}$$

For example, we can write $\bar{\mathtt{D}}_{\mathrm{pc}} = \frac{\mathtt{p\bar{D}}_{\mathrm{msc}}}{2}(1+\frac{\mathtt{p\bar{D}}_{\mathrm{msc}}}{2})\mathtt{p\bar{D}}_{\mathrm{ln}}$. We can also combine (3.3), (4.1), and $\bar{\mathtt{D}}_{\mathrm{pc}} = \frac{m_0 t l}{\mathsf{N}}$ to write the interesting relation

$$\ell = \frac{\mathtt{p\bar{D}}_{\mathrm{ln}}}{2} t, \tag{4.3}$$

which must be satisfied by the parameters if the success rate of $\mathtt{p\bar{D}}_{\mathrm{ps}}$ is to be achieved.

We can now start our analysis of the online execution behavior of the $\mathtt{p\bar{D}}$ tradeoff with a technical lemma. The lemma will be justified in Section 4.3.

Lemma 4.1. *The probability for the inverse* **p** *of an inversion target* $\mathbf{h} = F(\mathbf{p})$ *not to be found until the i-th online iteration step of the* $\mathtt{p\bar{D}}$ *tradeoff is*

$$\left\{1+\frac{\mathtt{p\bar{D}}_{\mathrm{msc}}}{2}(1-e^{-\frac{i}{t}})\right\}^{-\mathtt{p\bar{D}}_{\mathrm{ln}}}.$$

Proof. Note that the $(i+1)$-th step creates a chain of length $i+1$, under our convention for the unknown answer to be included in stating the chain length. Consider the ℓ perfect DP matrices as having been aligned on their ending points. The correct answer **p** cannot be found until the i-th online iteration step of the $\mathtt{p\bar{D}}$ tradeoff if and only if none of these DP matrices contain the correct input **p** to the inversion target in all their columns of distances at most i from the ending points. The probability for this event to occur is

$$\left(1-\frac{\sum_{j=1}^{i}\overleftarrow{\bar{m}}_j}{\mathsf{N}}\right)^{\ell} = \exp\left(-\frac{\ell}{\mathsf{N}}\sum_{j=1}^{i}\overleftarrow{\bar{m}}_j\right), \tag{4.4}$$

where $\overleftarrow{\bar{m}}_j$ is the number of points that appear in a column of distance j from the ending points in a *perfect* DP matrix.

Noting that all points of a perfect DP matrix are distinct, we can count the points according to rows, rather than in columns, to write

$$\sum_{j=1}^{i} \overleftarrow{m}_j = \#\left(\begin{array}{c}\text{chains of length} > i\\ \text{in perfect DP matrix}\end{array}\right)\cdot i + \sum_{k=1}^{i}\#\left(\begin{array}{c}\text{chains of length } k\\ \text{in perfect DP matrix}\end{array}\right)\cdot k.$$

It is argued within the proof of Proposition 3.3 that a perfect DP matrix created from m_0 starting points is expected to contain

$$\overrightarrow{m}_0\Big(1-\frac{1}{t}\Big)^{k-1}\frac{1}{t}\Big\{1+\frac{\bar{\mathrm{pD}}_{\mathrm{msc}}}{2}\frac{\overleftarrow{m}_k}{m}(e^{\frac{k}{t}}-1)\Big\}^{-2}$$

chains of length k, and we can observe that

$$\#\left(\begin{array}{c}\text{chains of length} > i\\ \text{in perfect DP matrix}\end{array}\right) = m - \#\left(\begin{array}{c}\text{chains of length} \le i\\ \text{in perfect DP matrix}\end{array}\right).$$

Using these facts, we can write the formula

$$\sum_{j=1}^{i} \overleftarrow{m}_j = m\cdot i - \overrightarrow{m}_0\sum_{k=1}^{i}\Big(1-\frac{1}{t}\Big)^{k-1}\frac{i-k}{t}\Big\{1+\frac{\bar{\mathrm{pD}}_{\mathrm{msc}}}{2}\frac{\overleftarrow{m}_k}{m}(e^{\frac{k}{t}}-1)\Big\}^{-2},$$

and after applications of (3.3) and (3.6), this becomes

$$\begin{aligned}\sum_{j=1}^{i} \overleftarrow{m}_j &= m\cdot i - mt\Big(1+\frac{\bar{\mathrm{pD}}_{\mathrm{msc}}}{2}\Big)\int_0^{\frac{i}{t}} e^{-x}\Big(\frac{i}{t}-x\Big)\Big\{1+\frac{\bar{\mathrm{pD}}_{\mathrm{msc}}}{2}(1-e^{-x})\Big\}^{-2}dx\\ &= mt\frac{2}{\bar{\mathrm{pD}}_{\mathrm{msc}}}\ln\Big(1+\frac{\bar{\mathrm{pD}}_{\mathrm{msc}}}{2}(1-e^{-\frac{i}{t}})\Big).\end{aligned}$$

It now suffices to substitute this into (4.4) and then apply (4.3) to arrive at the claimed formula. $\square$

Above lemma helps us to obtain an online computational complexity, which consists of two components, that is, online chain creations and alarm resolving process.

Proposition 4.1. *For the* $\mathtt{p\bar{D}}$ *tradeoff, the expected number of F-iterations required to generate the online chains is*

$$
\begin{aligned}
t^2 &\int_0^\infty \frac{\mathtt{p\bar{D}}_{\mathrm{ln}}}{2}\Big\{1+\frac{\mathtt{p\bar{D}}_{\mathrm{msc}}}{2}(1-e^{-x})\Big\}^{-\mathtt{p\bar{D}}_{\mathrm{ln}}} e^{-x}\,dx && (4.5)\\
&= t^2 \times \frac{\mathtt{p\bar{D}}_{\mathrm{ln}}}{\mathtt{p\bar{D}}_{\mathrm{msc}}(\mathtt{p\bar{D}}_{\mathrm{ln}}-1)}\Big\{1-\Big(1+\frac{\mathtt{p\bar{D}}_{\mathrm{msc}}}{2}\Big)^{1-\mathtt{p\bar{D}}_{\mathrm{ln}}}\Big\}. && (4.6)
\end{aligned}
$$

Proof. The $(i+1)$-th iteration for any pre-computation table is executed if and only if its online chain had not yet reached a DP and none of the DP matrices contain the correct input to the inversion target in their columns at distances at most i from the ending points. Although the two events that were just mentioned are not strictly independent, since ℓ is quite large in any practical situation, using Lemma 4.1, we can state

$$
\Big(1-\frac{1}{t}\Big)^i \Big\{1+\frac{\mathtt{p\bar{D}}_{\mathrm{msc}}}{2}(1-e^{-\frac{i}{t}})\Big\}^{-\mathtt{p\bar{D}}_{\mathrm{ln}}} \tag{4.7}
$$

as a good approximation of the probability for these events to occur.

The number of function iterations under consideration can now be written as

$$
\begin{aligned}
&\ell\sum_0^\infty \Big(1-\frac{1}{t}\Big)^i \Big\{1+\frac{\mathtt{p\bar{D}}_{\mathrm{msc}}}{2}(1-e^{-\frac{i}{t}})\Big\}^{-\mathtt{p\bar{D}}_{\mathrm{ln}}}\\
&= \ell t \int_0^\infty e^{-x}\Big\{1+\frac{\mathtt{p\bar{D}}_{\mathrm{msc}}}{2}(1-e^{-x})\Big\}^{-\mathtt{p\bar{D}}_{\mathrm{ln}}} dx.
\end{aligned}
$$

The stated formula can be obtained by applying (4.3) and explicitly computing the definite integral. □

Unfortunately, based on experimental evidences, we claim that, when an alarm occurs at the i-th onlne chain walk associated with a perfect table under focus,

$$
t \times \mathtt{\bar{D}}_{\mathrm{cr}}\Big(1 + \frac{0.615\,\mathtt{p\bar{D}}_{\mathrm{msc}}^{\frac{25}{36}}\,(1+\mathtt{p\bar{D}}_{\mathrm{msc}})^{0.8}}{2+\mathtt{p\bar{D}}_{\mathrm{msc}}}\Big(1-e^{-\frac{2.495+3.538\ln(1+5\mathtt{p\bar{D}}_{\mathrm{msc}})}{20.749}(\frac{i}{t})^{1.1}}\Big)\Big)^2 \tag{4.8}
$$

F-invocations are required to resolve the alarm. We will later show in Section

4.3 that, this claim is quite credible for a sufficiently wide range of parameters m and t, which covers almost all parameter combinations of interest in a theoretical approach. As was emphasized in Section 3.1, we are not claiming formula (4.8) to be *correct* in any theoretical sense. Note that, it is an interesting fact that an alarm produced by very short online chain requires approximately $\bar{\mathtt{D}}_{\mathrm{cr}}t$ iterations to resolve the alarm. We know from (3.9) that $\bar{\mathtt{D}}_{\mathrm{cr}}t$ is an average chain length of a perfect DP matrix. This fact would seem to make sense, because the online chain record technique, described in Section 2.1.1, do not quite work on very short online chain so that the corresponding pre-computation chain in the perfect DP table is fully regenerated,

Proposition 4.2. *The online phase of the* $\mathtt{p\bar{D}}$ *tradeoff is expected to require*

$$t^2\,\frac{\mathtt{p\bar{D}}_{\mathrm{ln}}\bar{\mathtt{D}}_{\mathrm{cr}}}{2}\int_0^\infty \left\{1+\frac{\mathtt{p\bar{D}}_{\mathrm{msc}}}{2}(1-e^{-x})\right\}^{-\mathtt{p\bar{D}}_{\mathrm{ln}}} e^{-x}\,(1-e^{-\mathtt{p\bar{D}}_{\mathrm{msc}}x}) \tag{4.9}$$

$$\times\left(1+\frac{0.615\,\mathtt{p\bar{D}}_{\mathrm{msc}}^{\frac{25}{36}}\,(1+\mathtt{p\bar{D}}_{\mathrm{msc}})^{0.8}}{2+\mathtt{p\bar{D}}_{\mathrm{msc}}}\left(1-e^{-\frac{2.495+3.538\ln(1+5\mathtt{p\bar{D}}_{\mathrm{msc}})}{20.749}x^{1.1}}\right)\right)^2 dx \tag{4.10}$$

invocations of the one-way function in relation to resolving of possible alarms.

Proof. In a similar manner with the proof of [16, Lemma 3], let us focus on a single perfect DP table among ℓ perfect tables. We already know that the probability for the i-th online chain walk step of the table to be executed is written as (4.7), replacing i by $(i-1)$. From this fact, one can write the probability for the online chain of the table to reach a DP at the i-th online iteration of $\mathtt{p\bar{D}}$ tradeoff as

$$\left\{1+\frac{\mathtt{p\bar{D}}_{\mathrm{msc}}}{2}(1-e^{-\frac{i-1}{t}})\right\}^{-\mathtt{p\bar{D}}_{\mathrm{ln}}}\left(1-\frac{1}{t}\right)^{i-1}\frac{1}{t}. \tag{4.11}$$

Now, let us find a probability of an alarm to occur by the ending point DP of this online chain of length i. By definition of $\overleftarrow{m}_k$ and $\mathscr{D}_k$, this probability can be written as

$$\begin{aligned}
&\frac{\overleftarrow{m}_i}{|\mathscr{D}_i|}+\Big(1-\frac{\overleftarrow{m}_i}{|\mathscr{D}_i|}\Big)\frac{\overleftarrow{m}_{i-1}}{|\mathscr{D}_{i-1}|}+\Big(1-\frac{\overleftarrow{m}_i}{|\mathscr{D}_i|}\Big)\Big(1-\frac{\overleftarrow{m}_{i-1}}{|\mathscr{D}_{i-1}|}\Big)\frac{\overleftarrow{m}_{i-2}}{|\mathscr{D}_{i-2}|}\\
&+\cdots+\Big(1-\frac{\overleftarrow{m}_i}{|\mathscr{D}_i|}\Big)\Big(1-\frac{\overleftarrow{m}_{i-1}}{|\mathscr{D}_{i-1}|}\Big)\cdots\Big(1-\frac{\overleftarrow{m}_1}{|\mathscr{D}_1|}\Big)\frac{\overleftarrow{m}_0}{|\mathscr{D}_0|}. \qquad (4.12)
\end{aligned}$$

Details to derive this formula are explained below. Since the online chain met a DP, we're aware of how far each node of the online chain away from the DP, so that each online node should be treated as an element of a certain set $\mathscr{D}_k$ for some k, rather than an entire space $\mathscr{N}$. An alarm occurs if and only if there exists an unique k ($1 \le k \le i$) such that the online node, k iterations away from the online DP, meets one of chain in a non-perfect DP matrix for the first time. At this time, firstly merging online node must meet a pre-computed point in a column, which position is the same amount of iterations far away from the DPs. Furthermore, an each event for an online chain to meet a pre-computation chain in a non-perfect matrix at a particular position for the first time is independent on its merging position. Thus, a desired probability can be obtained by adding probabilities of those events to occur for all position. For each online node k iterations away from an online DP ($1 \le k \le i$), a probability for preceding online nodes not to merge with pre-computation chains in a non-perfect matrix is $(1-\frac{\overleftarrow{m}_i}{|\mathscr{D}_i|})(1-\frac{\overleftarrow{m}_{i-1}}{|\mathscr{D}_{i-1}|})\cdots(1-\frac{\overleftarrow{m}_{k+1}}{|\mathscr{D}_{k+1}|})$, and a probability of this online node to merge with non-perfect DP matrix is $\frac{\overleftarrow{m}_k}{|\mathscr{D}_k|}$, so that a probability for an online chain k iterations away from the online DP to merge with the non-perfect DP matrix for the first time is $(1-\frac{\overleftarrow{m}_i}{|\mathscr{D}_i|})(1-\frac{\overleftarrow{m}_{i-1}}{|\mathscr{D}_{i-1}|})\cdots(1-\frac{\overleftarrow{m}_{k+1}}{|\mathscr{D}_{k+1}|})\frac{\overleftarrow{m}_k}{|\mathscr{D}_k|}$. Combining these probabilities for all k ($1 \le k \le i$), one can arrive at the desired formula (4.12).

By (3.6) and (3.8), for all k, $\frac{\overleftarrow{m}_k}{|\mathscr{D}_k|}=\frac{\bar{\mathtt{pD}}_{\mathrm{msc}}}{t}$. Hence, (4.12) can be written as

$$\begin{aligned}
\sum_{k=0}^{i}\Big(1-\frac{\bar{\mathtt{pD}}_{\mathrm{msc}}}{t}\Big)^k\frac{\bar{\mathtt{pD}}_{\mathrm{msc}}}{t} &= 1-\Big(1-\frac{\bar{\mathtt{pD}}_{\mathrm{msc}}}{t}\Big)^{i+1}\\
&\approx 1-\exp\Big(-\bar{\mathtt{pD}}_{\mathrm{msc}}\frac{i+1}{t}\Big). \qquad (4.13)
\end{aligned}$$

Combining this formula with (4.11), a probability of an alarm to occur at the i-th online chain walk step of a perfect table under focus is

$$\Big\{1+\frac{\mathtt{p}\bar{\mathtt{D}}_{\mathrm{msc}}}{2}(1-e^{-\frac{i-1}{t}})\Big\}^{-\mathtt{p}\bar{\mathtt{D}}_{\mathrm{ln}}}\Big\{1-\exp\Big(-\mathtt{p}\bar{\mathtt{D}}_{\mathrm{msc}}\frac{i+1}{t}\Big)\Big\}e^{-\frac{i-1}{t}}\frac{1}{t}. \qquad (4.14)$$

Still, events corresponding to (4.11) and (4.12) are not strictly independent, since ℓ is quite large in any practical situation, we can multiply two probabilities with negligible error. Accuracy of the formula (4.14) will be verified experimentally in Section 4.3.

Now, multiplying (4.8) to this, adding that for all i and then changing into an integral form, we can obtain the expected number of F invocations to resolving a possible alarm associated with a perfect table under focus as the following formula

$$t\,\bar{\mathtt{D}}_{\mathrm{cr}}\int_0^\infty\Big\{1+\frac{\mathtt{p}\bar{\mathtt{D}}_{\mathrm{msc}}}{2}(1-e^{-x})\Big\}^{-\mathtt{p}\bar{\mathtt{D}}_{\mathrm{ln}}}e^{-x}(1-e^{-\mathtt{p}\bar{\mathtt{D}}_{\mathrm{msc}}x})$$
$$\times\Big(1+\frac{0.615\,\mathtt{p}\bar{\mathtt{D}}_{\mathrm{msc}}^{\frac{25}{36}}(1+\mathtt{p}\bar{\mathtt{D}}_{\mathrm{msc}})^{0.8}}{2+\mathtt{p}\bar{\mathtt{D}}_{\mathrm{msc}}}\Big(1-e^{-\frac{2.495+3.538\ln(1+5\mathtt{p}\bar{\mathtt{D}}_{\mathrm{msc}})}{20.749}x^{1.1}}\Big)\Big)^2dx.$$

Finally, we can arrive at the desired formula by multiplying ℓ to this and applying (4.3) □

Now, combining Proposition 4.1 and Proposition 4.2, we can obtain the time memory tradeoff coefficient $\mathtt{p}\bar{\mathtt{D}}_{TM^2}$, the measure of how efficient tradeoff algorithm is, for the $\mathtt{p}\bar{\mathtt{D}}$ tradeoff.

Theorem 4.1. *The time memory tradeoff curve for the* $\mathtt{p}\bar{\mathtt{D}}$ *tradeoff is* $TM^2 = \mathtt{p}\bar{\mathtt{D}}_{TM^2}\mathsf{N}^2$, *where the tradeoff coefficient is given by*

$$\mathtt{p}\bar{\mathtt{D}}_{TM^2}=\int_0^\infty\frac{\{\ln(1-\mathtt{p}\bar{\mathtt{D}}_{\mathrm{ps}})\}^2}{\bar{\mathtt{D}}_{\mathrm{cr}}^2}\frac{\mathtt{p}\bar{\mathtt{D}}_{\mathrm{ln}}}{2}\Big\{1+\frac{\mathtt{p}\bar{\mathtt{D}}_{\mathrm{msc}}}{2}(1-e^{-x})\Big\}^{-\mathtt{p}\bar{\mathtt{D}}_{\mathrm{ln}}}e^{-x}$$
$$\times\left(\begin{array}{r}1+\bar{\mathtt{D}}_{\mathrm{cr}}(1-e^{-\mathtt{p}\bar{\mathtt{D}}_{\mathrm{msc}}x})\Big(1+\dfrac{0.615\,\mathtt{p}\bar{\mathtt{D}}_{\mathrm{msc}}^{\frac{25}{36}}(1+\mathtt{p}\bar{\mathtt{D}}_{\mathrm{msc}})^{0.8}}{2+\mathtt{p}\bar{\mathtt{D}}_{\mathrm{msc}}}\\ \times\Big(1-e^{-\frac{2.495+3.538\ln(1+5\mathtt{p}\bar{\mathtt{D}}_{\mathrm{msc}})}{20.749}x^{1.1}}\Big)\Big)^2\end{array}\right)dx.$$

Proof. The tradeoff curve of the $\mathtt{p}\bar{\mathtt{D}}$ tradeoff is direct consequences of Proposition 4.1, Proposition 4.2 and the fact that the pre-computation tables contain $M = m\ell$ entries in total. In order to derive the claimed formula, one can use the relation $\frac{mt\ell}{\mathsf{N}} = \frac{-\ln(1-\mathtt{p}\bar{\mathtt{D}}_{\mathrm{ps}})}{\bar{\mathtt{D}}_{\mathrm{cr}}}$ by (3.3) and (4.1). □

We emphasize that Proposition 4.2 and Theorem 4.1 rely on the empirical result (4.8). Hence, both claims should be understood as practical formulas, not theoretically correct formulas.

4.2 Storage Optimization

As for any other DP variants algorithm, the effects of ending point truncation are need to be discussed. However, since the analysis is very similar to that of the perfect DP tradeoff, we quickly review Section 3.1.2 and omit the detailed explanation.

The effect of truncating ending points on increasing pre-computation cost to produce a perfect table is identical with that of the perfect DP tradeoff, so that Lemma 3.6 and its ignorable level, presented below the lemma, can be applied to the case of the $\mathtt{p}\bar{\mathtt{D}}$ tradeoff. Moreover, in a similar way to prove Lemma 3.7, one can compute the number of extra one-way function invocations induced by truncation-related alarms for the $\mathtt{p}\bar{\mathtt{D}}$ tradeoff.

Lemma 4.2. *Consider a truncation map with a truncated space of size r. Assume that the truncated space is much smaller than the DP space and that r has been chosen to be large enough for the occurrences of indistinguishable ending points caused by truncations to be sufficiently limited. Then the number of extra one-way function invocations induced by truncation-related alarms is expected to be*

$$t^2 \frac{m}{r} \frac{\mathtt{p}\bar{\mathtt{D}}_{\mathrm{ln}}}{\mathtt{p}\bar{\mathtt{D}}_{\mathrm{msc}}} \ln\left(1+\frac{\mathtt{p}\bar{\mathtt{D}}_{\mathrm{msc}}}{2}\right) \int_0^\infty \left\{1+\frac{\mathtt{p}\bar{\mathtt{D}}_{\mathrm{msc}}}{2}(1-e^{-x})\right\}^{-\mathtt{p}\bar{\mathtt{D}}_{\mathrm{ln}}} e^{-(1+\mathtt{p}\bar{\mathtt{D}}_{\mathrm{msc}})x}\,dx,$$

during the full processing of the $\mathtt{p}\bar{\mathtt{D}}$ *tradeoff.*

Proof. Let us quickly sketch a proof. The proof is very similar with the proof of Lemma 3.7. The probability for an online chain of a perfect table under focus to become a DP chain of length i and not merge into the perfect DP matrix, but have a truncated ending point that coincides with a truncated ending point can be written as

$$\Big\{1+\frac{\bar{\mathtt{pD}}_{\mathrm{msc}}}{2}(1-e^{-\frac{i-1}{t}})\Big\}^{-\bar{\mathtt{pD}}_{\mathrm{ln}}}\Big(1-\frac{1}{t}-\frac{|\mathtt{DM}|}{\mathtt{N}}\Big)^{i-1}\Big(\frac{1}{t}-\frac{m}{\mathtt{N}}\Big)\frac{m}{r},$$

when a truncated space is of size r. Recall that the expression $\big\{1+\frac{\bar{\mathtt{pD}}_{\mathrm{msc}}}{2}(1-e^{-\frac{i-1}{t}})\big\}^{-\bar{\mathtt{pD}}_{\mathrm{ln}}}$ from Lemma 4.1 presents the probability for the algorithm not to return the correct answer until $(i-1)$-th online iteration. Summation of this over all i can be interpreted as the definite integral unless t is extremely small, and it presents the probability for an online chain creation of a perfect table under focus to produce a truncation-related alarm.

As was the perfect DP tradeoff, a truncation-related alarm requires iterations of the one-way function to resolve this false alarm as much as the average length of chains in a perfect DP matrix.

Multiplying the above formula with the average chain length (3.9) and the number of tables ℓ, and then combining this with (4.3), we arrive at the claim. $\square$

Now, one can realize that the number of extra one-way function invocations induced by truncation-related alarms, when the $\bar{\mathtt{pD}}$ tradeoff is fully processed, is of order $t^2\frac{m}{r}\cdot\Theta(1)$.

Recall that the sum of formulas in Proposition 4.1 and Proposition 4.2 expresses the normal expected number of F-iterations required for execution of the online phase of the $\bar{\mathtt{pD}}$ tradeoff before ending point truncation. The next proposition expresses the effects of ending point truncation on the total online time. For example, with $\bar{\mathtt{pD}}_{\mathrm{msc}}=2$ and $r=2^5m$, if assume that $\ell=3.32t$ is used to achieve 99% success probability at this $\bar{\mathtt{pD}}_{\mathrm{msc}}$ value, the expected numbers of normal iterations and truncation-related iterations become $0.734478t^2$ and $0.276592\,\frac{1}{32}t^2$, respectively. Thus, the ending point truncation increases the number of one-way function iterations by a mere 1.1%.

Proposition 4.3. *Suppose that the online phase of a* $\bar{\mathtt{pD}}$ *tradeoff implementation that stores each ending point in full requires* T *iterations of the one-way function to complete. Consider a truncation map for which the truncated space is of size* $r = 2^{\varepsilon}m \ll \mathsf{N}$. *If* ε *is large enough for the occurrences of indistinguishable ending points caused by truncations to be ignored, then the implementation with the ending point truncation requires*

$$\frac{\frac{\bar{\mathtt{pD}}_{\mathrm{ln}}}{\bar{\mathtt{pD}}_{\mathrm{msc}}} \ln\left(1+\frac{\bar{\mathtt{pD}}_{\mathrm{msc}}}{2}\right) \int_0^\infty \left\{1+\frac{\bar{\mathtt{pD}}_{\mathrm{msc}}}{2}(1-e^{-x})\right\}^{-\bar{\mathtt{pD}}_{\mathrm{ln}}} e^{-(1+\bar{\mathtt{pD}}_{\mathrm{msc}})x}\,dx}{\frac{1}{t^2}((4.5)+(4.9))} \frac{T}{2^{\varepsilon}}$$

additional iterations of the one-way function to complete.

For parameters satisfying $\bar{\mathtt{pD}}_{\mathrm{msc}} = 2$ and $\ell = 3.32t$, the above is $0.376583\frac{T}{2^{\varepsilon}}$. This implies that, for parameters of interest, a small ε is enough to keep the negative effects of ending point truncation on the online time to a reasonably small level.

In all, the starting point can be stored using slightly more than $\log m$ bits. Ending point DPs can be truncated so that a little more than $\log m$ bits of information is retained with very little negative effect on the success probability, pre-computation cost, and online time. The index file technique can be used to remove almost $\log m$ further bits per ending point without any loss of information. In conclusion, storage of each starting point and ending point pair requires a little more than $\log m$ bits. This was also the conclusion obtained for the previous two DP tradeoff algorithms and the original DP tradeoff, analyzed in [18].

4.3 Experiment Results

We have verified the correctness of major parts of our complexity analysis with experiments. Throughout this section, we use the modified version of the MD5 hash function, that was described in Section 3.1.3, as the one-way function. We always set the moderately large chain length bound of $\hat{t} = 15t$.

The first experiment was designed to verify (4.7) and (4.14) simultaneously. Recall that both are deeply related to computing the online complexity of the $\bar{\mathtt{pD}}$ tradeoff. More precisely, replacing i in (4.7) by $i-1$, it presents a probability for the i-th online chain walk to be executed, and (4.14) presents a probability of alarm to occur at the i-th online iteration of a pre-computation perfect table under focus, during the fully processing of the $\bar{\mathtt{pD}}$ tradeoff.

We have observed when $\mathsf{N} = 2^{35}$ and 2^{37}. After fixing suitable parameters $\vec{m}_0$, t, and ℓ, which directly determines $\bar{\mathtt{pD}}_{\text{ln}} = \frac{2\ell}{t}$, we can expect the average number of entries in a perfect table from (3.3), and hence $\bar{\mathtt{pD}}_{\text{msc}}$. We first generated ℓ perfect DP tables from $\vec{m}_0$ starting points. Then we generated suitably many inversion targets for each parameter set, and for each inversion target, a entire $\bar{\mathtt{pD}}$ tradeoff, described in Section 2.1.1, was processed. During the process, we separately counted and recorded, whenever a single chain walk to create an online chain of a perfect table invoked and its iteration produced alarm, together with corresponding online chain length i ($1 \leq i \leq \hat{t}$). These first and second records divided by $\ell\times$ (# of targets) are taken as test values which verify (4.7) and (4.14), respectively.

The test results are provided by Figure 4.1 and Figure 4.2 for various parameter sets. For each graph box in Figure 4.1, the probability (y-axis) for the i-th online chain walk of a pre-computation table under focus to be preformed during a full process of the $\bar{\mathtt{pD}}$ tradeoff is given for each chain length i (x-axis). Lines correspond to our theory, as given by (4.7), and dots represent the counted test value associated with each chain length, obtained through tests, divided by number of

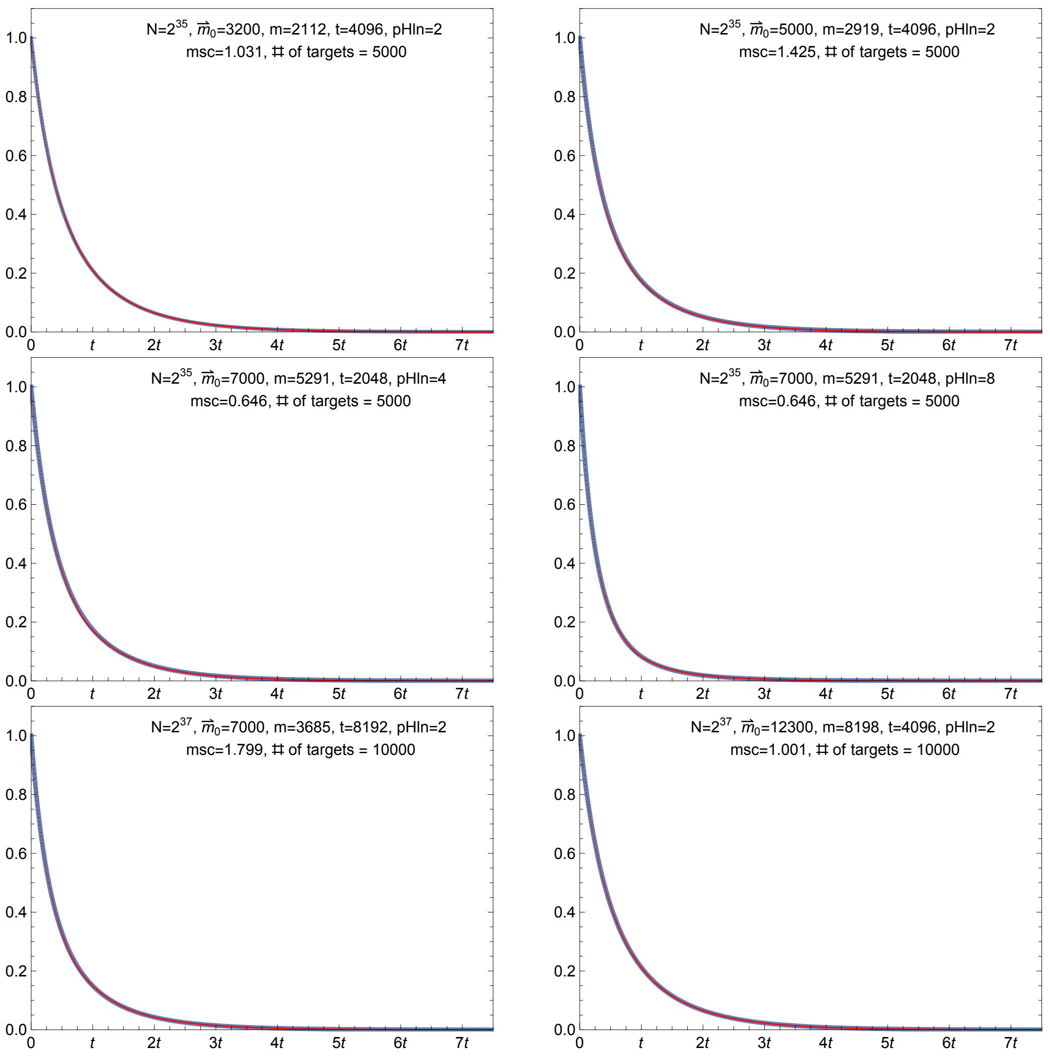

Figure 4.1: Probability for the *i*-th online chain walk of a pre-computation table during the entire process of the $\mathtt{p\bar{D}}$ tradeoff to be executed (test:*dots*; theory:*line*; $\hat{t} = 15t$).

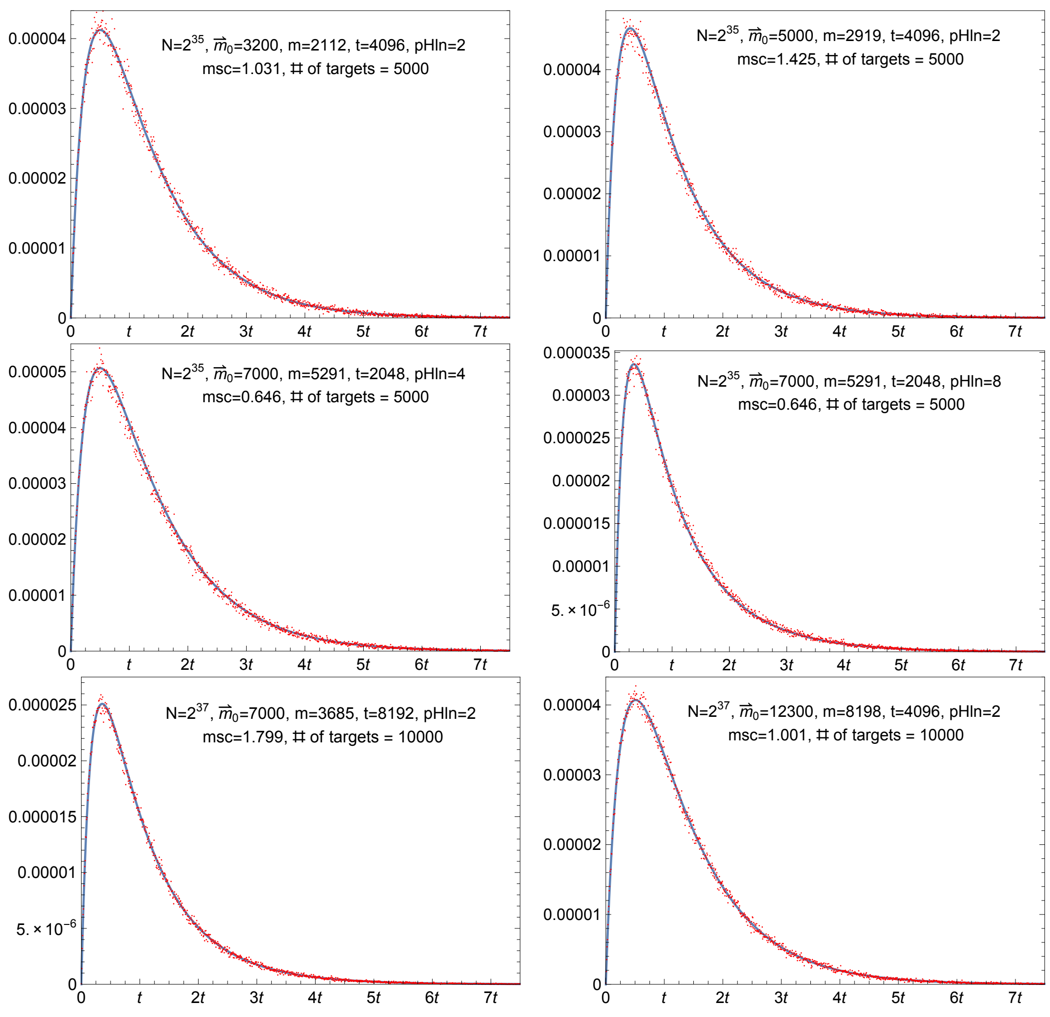

Figure 4.2: The probability of alarm to occur at the *i*-th iteration of a precomputation table during the entire process of the $\bar{\text{p}\text{D}}$ tradeoff (test:*dots*; theory:*line*; $\hat{t} = 15t$).

tables ℓ. For each graph box in Figure 4.2, the probability (y-axis) of occurring alarm at the i-th iteration of a table under focus during the entire process of the $\bar{\texttt{pD}}$ tradeoff is given for each chain length i (x-axis). Lines correspond to our theory, as given by (4.14), and the corresponding count ratios obtained through tests are plotted as dots. Even though our chain length bound was $\hat{t} = 15t$, we have displayed the data only for chain lengths less than approximately $7t$. Furthermore, in each box, we only plotted approximately 1000 dots, since densely packing all $7t$ dots into each box made the graphs harder to comprehend.

The experimental data agrees well with our theory in all the boxes in Figure 4.1 and Figure 4.2. Notice that the test results are less reliable at the large chain lengths. This is because longer DP chains appeared less frequently and these large chain length data were obtained from a smaller number of chains as like for the perfect DP tradeoff in Section 3.1.3. A much larger number of DP matrices would need to be generated to obtain meaningful test values at lengths much larger than $7t$.

Our final experiment measured the cost of regenerating the pre-computation chain to deal with an alarm, produced by an online chain of each length, and we claimed that the test values are close to our prediction formula (4.8). Likewise, the modified version of the MD5 hash function is also regarded as an one-way function for this experiments.

After choosing $\vec{m}_0$ and t, we created multiple perfect DP tables from $\vec{m}_0$ starting points. Then from (3.3), we can expect the number of table entry m, consisted in a perfect DP matrix, and so can $\bar{\texttt{pD}}_{\text{msc}}$. For each pre-computation table, we generated as many online chains as was required to observe a sufficiently large number of alarms. For each merge, the associated pre-computation chain was generated, up to the point of merge, that is referred to as the online chain records, previously explained in Section 2.1.1, and the length of this regenerating chain segment was recorded.

This process looks familiar with the last experiments done in Section 3.1.3.

However, in these experiments, when an alarm occurs, the length of the regenerated pre-computation chain segment was recorded associated with length of the online chain that produced the alarm. Also, the number of alarms was counted, together with its online chain length. Then, the ratios obtained by dividing the first cost-related record by the second count-related record, associated with each online chain length are treated as our test value for verifying (4.8).

Our test results for each various parameter set, together with the predictions given by formula (4.8), are provided in Figure 4.3. Even though our chain length bound was $\hat{t} = 15t$, we have displayed the data only for chain lengths less than $8t$. A reasonable excuses for this will be presented later in this section. In each box, we only plotted 1000 dots, since densely packing all $8t$ dots into each box made the graphs harder to comprehend.

In each graph box in Figure 4.3, the number of one-way function iterations (y-axis), divided by t, to resolve an alarm that occurs at each online chain walk step (x-axis) is given. In addition, numbers of inversion targets and perfect DP tables to be generated for each parameter set can be found. Each line plotted through formula (4.8) is close to the average number of one-way function iterations required per alarm associated to each online chain length that was obtained experimentally. One can believe that formula (4.8) will be quite reliable, at least for all parameter choices satisfying $0 < \bar{\mathrm{pD}}_{\mathrm{msc}} \leq 2.4$. However, it is possible that one can find another even closer formula than ours (4.8).

As an additional justification for our claimed formula (4.8), one can newly compute the number of F-invocations in relation to the resolving of a possible alarm in a single perfect DP table, not considering parallel treatment of pre-computation tables. Recall that it was already claimed in Proposition 3.4, which accuracy was verified in Section 3.1.3. Combining the probability of an online chain to meet a DP at the i-th iteration, $(1-\frac{1}{t})^{i-1}\frac{1}{t}$ with (4.13), the re-claimed cost can be written as

$$t\,\bar{\mathtt{D}}_{\mathrm{cr}} \int_0^\infty e^{-x}\left(1-e^{-\mathtt{p}\bar{\mathtt{D}}_{\mathrm{msc}}x}\right) \times \left(1+\frac{0.615\,\mathtt{p}\bar{\mathtt{D}}_{\mathrm{msc}}^{\frac{25}{36}}\left(1+\mathtt{p}\bar{\mathtt{D}}_{\mathrm{msc}}\right)^{0.8}}{2+\mathtt{p}\bar{\mathtt{D}}_{\mathrm{msc}}}\left(1-e^{-\frac{2.495+3.538\ln(1+5\mathtt{p}\bar{\mathtt{D}}_{\mathrm{msc}})}{20.749}x^{1.1}}\right)\right)^2 dx. \tag{4.15}$$

Thus, we claim that the above expression (4.15) presents the same value as the previously justified formula (3.11). Two formulas are plotted for matrix stopping constant $\mathtt{p}\bar{\mathtt{D}}_{\mathrm{msc}}$ in Figure 4.4, simultaneously. In addition, for various $\mathtt{p}\bar{\mathtt{D}}_{\mathrm{msc}}$, calculated values from those two formulas are presented in Table 4.1. These two comparisons between (3.11) and (4.15) assure us that claimed formula (4.8) can be quite reliable at least for all parameter choices satisfying $0 < \mathtt{p}\bar{\mathtt{D}}_{\mathrm{msc}} \leq 4.0$.

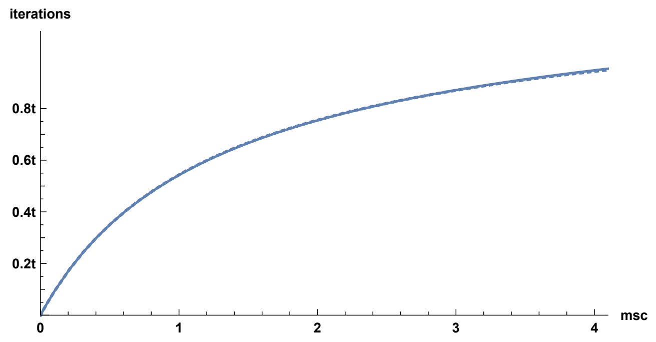

Figure 4.4: The number of one-way function iterations in relation to the resolving of a possible alarm in a single perfect DP table(Proposition 3.4: *dots*; (4.15): *line*).

Finally, let us explain reasons of plotting each graph in Figure 4.3 only for chain lengths less than $8t$ to be acceptable. Long online DP chains, further, alarms produced by long online chains appear less frequently by the already known probability $(1-e^{-\mathtt{p}\bar{\mathtt{D}}_{\mathrm{msc}}\frac{i+1}{t}})e^{-\frac{i-1}{t}}\frac{1}{t}$ from (4.13) Thus, the test results are less reliable at the large chain lengths. Nevertheless, our test results of large chain length over $5t$ are plotted in Figure 4.5, together with (4.8). As explained, since the number of alarms related to long chains is very small, test values are scattered, but one can

Table 4.1: The number of one-way function iterations in relation to the resolving of a possible alarm in a single perfect DP table for various $\mathrm{p}\bar{\mathrm{D}}_{\mathrm{msc}}$.

$\mathrm{p}\bar{\mathrm{D}}_{\mathrm{msc}}$	(4.15)	Proposition 3.4	$\dfrac{(4.15)}{\text{Proposition 3.4}}$
	$\frac{1}{t} \times \#(\mathrm{itr})$	$\frac{1}{t} \times \#(\mathrm{itr})$	
0.2	0.17062	0.17052	1.00057
0.5	0.35026	0.35047	0.99941
0.8	0.47676	0.47737	0.99873
1.1	0.57138	0.57234	0.99833
1.4	0.64525	0.64641	0.99820
1.7	0.70478	0.70597	0.99832
2.0	0.75397	0.75500	0.99865
2.3	0.79545	0.79611	0.99916
2.6	0.83099	0.83112	0.99984
2.9	0.86188	0.86132	1.00065
3.2	0.88904	0.88764	1.00158
3.5	0.91317	0.91080	1.00261
3.8	0.93481	0.93134	1.00372
4.0	0.94804	0.94380	1.00450

expect that the desired cost would maintain a certain constant value for sufficiently large online chain lengths. Our claimed formula (4.8) also has these properties and each line in Figure 4.5 looks very similar to the test dots. But still, the formula is doubtable.

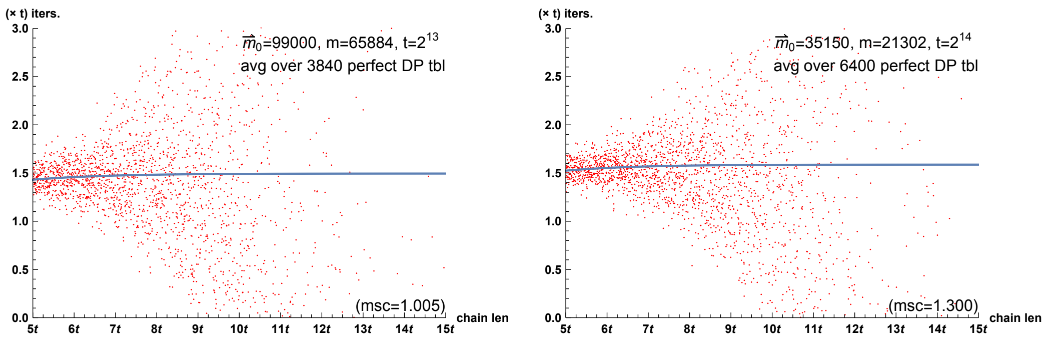

Figure 4.5: The number of one-way function iterations required per each alarm that occurs at the i-th online chain walk of a perfect DP table (test:*dots*; theory:*line*; $\mathsf{N} = 2^{42}$; $\hat{t} = 15t$; $5t \leq i \leq 15t$).

However, we claim that the cost for resolving possible alarms produced by *sufficiently long* online chain has negligible effect on the entire cost required to resolve alarms. More precisely, alarms produced by long online chains of length over $8t$ are very rare, as seen in Figure 4.2 and (4.14), so that the corresponding alarm-related cost can be ignored. In the Table 4.2, for various success requirements and matrix stopping constants, the ratios of cost for resolving a possible alarm produced by online chain of length less than $8t$, that is $\sum_{i=1}^{8t}(4.14)\times(4.8)$, to total alarm-related cost, that is $\sum_{i=1}^{\infty}(4.14)\times(4.8)$, are presented. At this time, the alarm-related cost is focused on amount of requirements for the process of a single fixed perfect DP table during a full execution of the $\bar{\mathtt{pD}}$ tradeoff.

Moreover, one can easily plot the formulas $\sum_{i=1}^{8t}(4.14)\times(4.8)$ and $\sum_{i=1}^{\infty}(4.14)\times(4.8)$ as functions of $\bar{\mathtt{pD}}_{\text{msc}}$, for each fixed success rate. We omit those graphs, because two plotted lines do agree each other, so that it is hard to distinguish those for every success probability. Thus, we can safely conclude that the cost for resolving a possible alarm produced by long online chain over $8t$ during a full execution of the $\bar{\mathtt{pD}}$ tradeoff, can be ignored, so that it is enough to verify the accuracy of the formula (4.8) only when $1 \leq i \leq 8t$.

Table 4.2: The ratios of cost for resolving a possible alarm produced by online chain of length less than $8t$ to total alarm-related cost for various $\bar{\mathtt{pD}}_{\mathrm{ps}}$ and $\bar{\mathtt{pD}}_{\mathrm{msc}}$, focusing on the process of a single perfect DP table.

$\bar{\mathtt{pD}}_{\mathrm{ps}}$	$\bar{\mathtt{pD}}_{\mathrm{msc}}$	$\frac{\sum_{i=1}^{8t}(4.14)\times(4.8)}{\sum_{i=1}^{\infty}(4.14)\times(4.8)}$	$\bar{\mathtt{pD}}_{\mathrm{ps}}$	$\bar{\mathtt{pD}}_{\mathrm{msc}}$	$\frac{\sum_{i=1}^{8t}(4.14)\times(4.8)}{\sum_{i=1}^{\infty}(4.14)\times(4.8)}$
25%	0.2	0.99826	50%	0.2	0.99845
	0.7	0.99903		0.7	0.99914
	1.2	0.99920		1.2	0.99929
	1.7	0.99926		1.7	0.99933
	2.2	0.99928		2.2	0.99936
	2.7	0.99929		2.7	0.99937
	3.2	0.99930		3.2	0.99937
	3.7	0.99930		3.7	0.99937
	4.2	0.99930		4.2	0.99937
	4.7	0.99930		4.7	0.99937
90%	0.2	0.99910	95%	0.2	0.99933
	0.7	0.99951		0.7	0.99963
	1.2	0.99960		1.2	0.99970
	1.7	0.99962		1.7	0.99972
	2.2	0.99963		2.2	0.99973
	2.7	0.99964		2.7	0.99973
	3.2	0.99963		3.2	0.99972
	3.7	0.99963		3.7	0.99972
	4.2	0.99963		4.2	0.99972
	4.7	0.99962		4.7	0.99971
99%	0.2	0.99970	99.9%	0.2	0.99993
	0.7	0.99984		0.7	0.99996
	1.2	0.99987		1.2	0.99997
	1.7	0.99987		1.7	0.99997
	2.2	0.99988		2.2	0.99997
	2.7	0.99988		2.7	0.99997
	3.2	0.99987		3.2	0.99997
	3.7	0.99987		3.7	0.99997
	4.2	0.99987		4.2	0.99997
	4.7	0.99986		4.7	0.99997

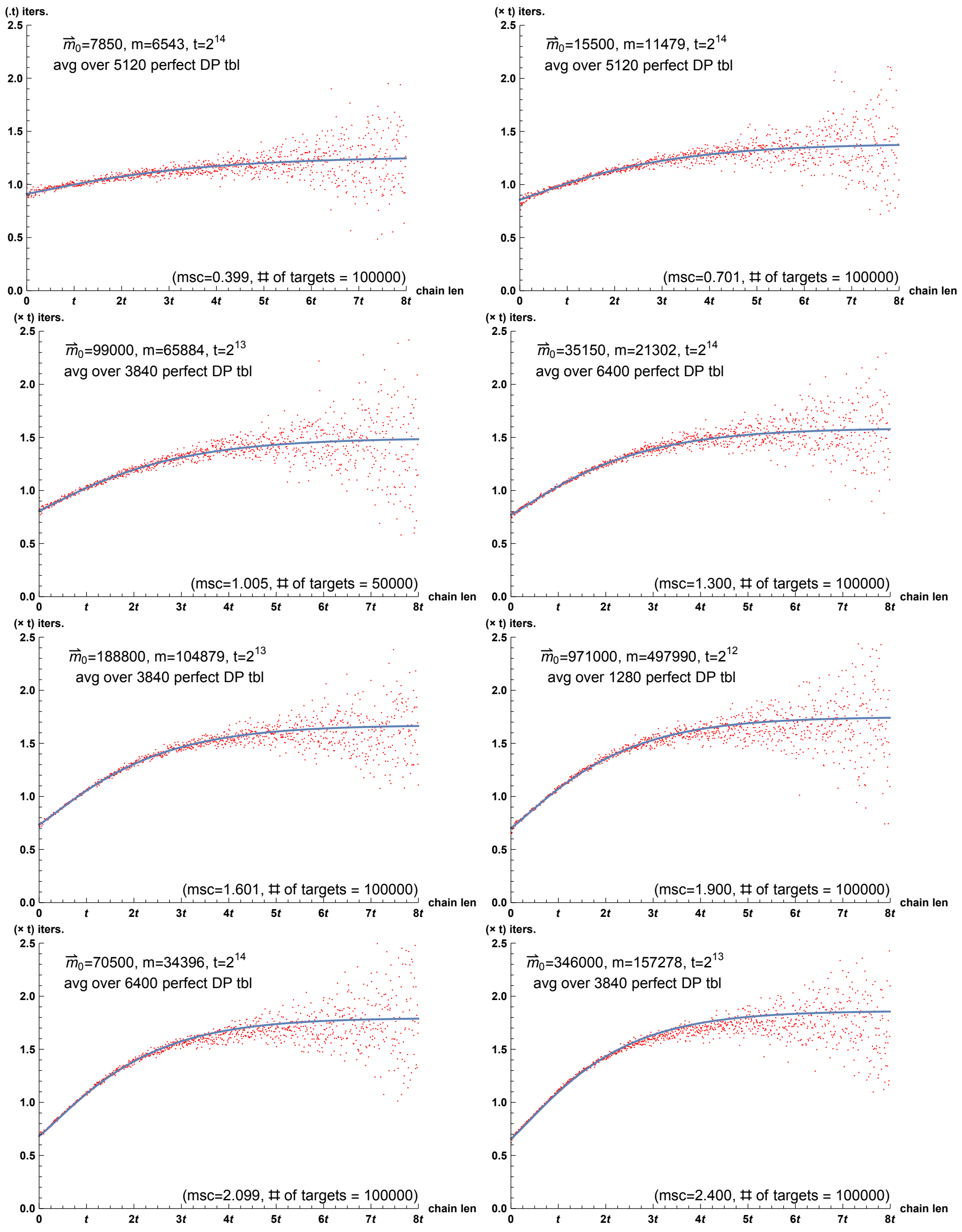

Figure 4.3: The number of one-way function iterations required per each alarm that occurs at the *i*-th online chain walk of a perfect DP table (test:*dots*; theory:*line*; $\mathsf{N} = 2^{42}$; $\hat{t} = 15t$).

Chapter 5

Comparisons Focused on Theoretical Complexities

Formulas for the two DP variants, that are the perfect DP, and perfect parallel DP tradeoffs, and the perfect rainbow tradeoff, which give the success rates, online efficiencies, and pre-computation costs in terms of the algorithm parameters, were obtained in the previous two chapters. Corresponding formulas for the original DP, parallel DP, and non-perfect rainbow tradeoffs were provided in earlier works [18] and [16], and were briefly reviewed in Section 2.3. In this chapter, we gather all of these information to compare the performances of the four mentioned DP tradeoff algorithms, and then compare the most efficient DP tradeoff against the perfect and non-perfect rainbow tradeoffs.

We exclude the classical Hellman tradeoff from our comparisons. It was mainly due to its performance being very similar to that of the non-perfect DP tradeoff. Anyone who read this thesis very carefully can compare the performance of the classical Hellman tradeoff with those of other tradeoff algorithms, using the information from the previous work [18]. However, we're already aware of the fact that the original DP and the classical Hellman tradeoffs present similar performance.

5.1 Method of Comparison

Let us describe the method to be used in comparing the performances of different tradeoffs. The approach we will use in this section was firstly suggested by [18].

Note that a time memory tradeoff method can fail to return the correct answer to the given inversion problem and that any tradeoff method is sure to require less resources if it is allowed to operate at a lower success rate. Hence, a fair performance comparison of the tradeoff methods must compare their various execution complexities under parameters for each tradeoff that correspond to a common success rate.

One can accept the tradeoff coefficient $\frac{TM^2}{\mathsf{N}^2}$ as providing a good measure of how efficient a tradeoff method is during the online phase, with a smaller value indicating a better method. Indeed, if Method-A has a smaller tradeoff coefficient than Method-B, then Method-A is expected to require a smaller online time T in solving an inversion problem than Method-B, when the two are provided with pre-computation tables of equal storage complexity M. Furthermore, since each of the six tradeoff methods we are comparing allows for tradeoffs between time and memory of the same $TM^2 = c \cdot \mathsf{N}^2$ form for some constant c, comparison of their tradeoff coefficients can be understood to be a *simultaneous* comparison of the online time T at all possible choices of the storage complexity M.

Although we have stated that the tradeoff coefficient is an accurate measure of the online efficiency of a tradeoff method, certain adjustments must be made before we can make comparisons of different tradeoff methods based on their tradeoff coefficients. Recall that the storage complexity M that was used in computing our tradeoff coefficients was the total number of entries written to the pre-computation tables, but the physical number of bits required to store each table entry actually depended on the tradeoff method and its parameters. As an example, suppose that we were given parameters for Method-A and Method-B with which the two methods would call for roughly comparable online resources, but which would required Method-A and Method-B to allocate 10 bits and 20 bits, respectively, to each pre-computation table entry. Then, to be fair, one must compare the $\frac{100TM^2}{\mathsf{N}^2}$ value computed for Method-A against the $\frac{400TM^2}{\mathsf{N}^2}$ value computed for Method-B, or, equivalently, compare Method-A's $\frac{1}{4}\frac{TM^2}{\mathsf{N}^2}$ against Method-B's $\frac{TM^2}{\mathsf{N}^2}$. In other words, the comparison of online efficiencies must be made between *adjusted tradeoff coefficients* that account for relative differences in the number of bits allocated to each table entry by the different tradeoff methods.

The set of relative adjustments to the tradeoff coefficients that is most appropriate will be different for every situation, and the precise adjustment factors become available only after one fixes the parameters and decides on how aggressively to apply the many storage reduction techniques. The previous work [18] provided a careful discussion with examples as to how the relative adjustments of the tradeoff coefficients are to be carried out in practice.

So far, we have explained that the tradeoff methods need to be compared under parameters achieving a common success rate and that the adjusted tradeoff coefficients allow for direct comparisons of the online efficiencies of different tradeoff methods. Now, note that if two tradeoff methods present the same online efficiency at the same success rate, one would prefer to use the one with a smaller pre-computation cost. That is, a fair comparison of tradeoff *performances* must also account for the cost of pre-computation. It is clear that the pre-computation coefficient can be used to presents this cost directly.

One can expect a tradeoff method to behave more efficiently after a larger investment in pre-computation. However, an amount of pre-computation which can be invested depends on each implementer's environment, so it is very difficult to decide optimal tradeoff parameters for every situation. The solution is to draw a pre-computation coefficient versus adjusted tradeoff coefficient curve for each tradeoff method. Each curve will be a concise visual display of what level of online efficiency is reachable by a tradeoff method after a certain amount of pre-computation effort. The implementer can decide which tradeoff is better for the specific situation he or she is faced with after viewing the whole range of options made available by the different tradeoff methods.

5.2 Comparison of DP Variants

Since we are going to compare the tradeoff methods at a few common fixed probabilities of success, the symbols $\mathtt{D}_{\text{ps}}$, $\mathtt{pD}_{\text{ps}}$, $\bar{\mathtt{D}}_{\text{ps}}$, and $\mathtt{p}\bar{\mathtt{D}}_{\text{ps}}$ will be now treated as fixed constants. We will use a notation $\mathtt{X}$, which can be replaced by any of $\mathtt{D}$, $\mathtt{pD}$, $\bar{\mathtt{D}}$, and $\mathtt{p}\bar{\mathtt{D}}$.

Let us explain how one may plot the pre-computation coefficient versus (adjusted) tradeoff coefficient curves for the four DP tradeoff algorithms. Throughout this chapter, we will refer to this curve simply as the *pc-tc curve*.

The pre-computation coefficients $\mathtt{D}_{\text{pc}}$ and $\bar{\mathtt{D}}_{\text{pc}}$ for the non-perfect DP and perfect DP variants, respectively, are regarded as functions of the single parameter $\mathtt{D}_{\text{msc}}$, $\mathtt{pD}_{\text{msc}}$, $\bar{\mathtt{D}}_{\text{msc}}$, and $\mathtt{p}\bar{\mathtt{D}}_{\text{msc}}$, for each DP tradeoffs, from (2.2) and (4.1), when a fixed success rate is required. Similarly, every tradeoff coefficient $\mathtt{X}_{TM^2}$ of the DP variants can be seen as a function of the single parameter $\mathtt{X}_{\text{msc}}$ from (2.4), (2.5), Theorem 3.1, and Theorem 4.1, because both $\mathtt{D}_{\text{cr}}$ and $\bar{\mathtt{D}}_{\text{cr}}$ are functions of the matrix stopping constants, from (2.1) and Proposition 3.3. Thus, all the pc-tc curves for the DP variants may be drawn as curves parameterized by $\mathtt{X}_{\text{msc}}$.

It is important to understand that, even when the success rate $\mathtt{X}_{\text{ps}}$ and curve parameter $\mathtt{X}_{\text{msc}}$ are fixed to specific values, there still remains a single degree of freedom concerning the tradeoff algorithm parameters $m_{\mathtt{X}}$, $t_{\mathtt{X}}$, and $\ell_{\mathtt{X}}$, with which one can realize the tradeoff between the online time T and the storage requirement M. That is, the ability of the DP method to provide tradeoffs between online time and storage requirement is unimpaired by restrictions on the success rate and the matrix stopping constant.

To be more concrete for the perfect DP tradoeff, suppose that one is given specific $\bar{\mathtt{D}}_{\text{ps}}$ and $\bar{\mathtt{D}}_{\text{msc}}$ values, together with any T and M that satisfy the tradeoff curve, where the tradeoff coefficient $\bar{\mathtt{D}}_{TM^2}$ has been computed from the given $\bar{\mathtt{D}}_{\text{ps}}$ and $\bar{\mathtt{D}}_{\text{msc}}$ values by Theorem 3.1. Then it is easy to check that the sequentially defined parameter set

$$t_{\bar{\mathtt{D}}} = \left\{ \frac{\bar{\mathtt{D}}_{\text{msc}}\bar{\mathtt{D}}_{\text{cr}}}{\bar{\mathtt{D}}_{\text{ps}}} \left(1 + \frac{1+0.577\bar{\mathtt{D}}_{\text{msc}}}{1+0.451\bar{\mathtt{D}}_{\text{msc}}} \frac{\bar{\mathtt{D}}_{\text{msc}}}{1+\bar{\mathtt{D}}_{\text{msc}}} \right)^{-1} T \right\}^{\frac{1}{2}}, \tag{5.1}$$

$$m_{\bar{\mathtt{D}}} = \frac{\bar{\mathtt{D}}_{\text{msc}}\mathsf{N}}{t^2}, \tag{5.2}$$

$$\ell_{\bar{\mathtt{D}}} = \frac{\mathsf{N}}{mt\bar{\mathtt{D}}_{\text{cr}}} \left\{ -\ln(1-\bar{\mathtt{D}}_{\text{ps}}) \right\} \tag{5.3}$$

satisfies the four requests or restrictions on $\bar{\mathtt{D}}_{\text{ps}}$, $\bar{\mathtt{D}}_{\text{msc}}$, T, and M. The equivalence of (5.3) and (3.7) implies that the success rate $\bar{\mathtt{D}}_{\text{ps}}$ will be achieved with these parameters, while (5.2) ensures that the given $\bar{\mathtt{D}}_{\text{msc}}$ value is adhered to. Furthermore, since (5.1) and the first equation in the proof of Theorem 3.1 are equivalent, the online phase is expected to terminate in the requested time T. Finally, since both the storage requirement under the above parameter set and the requested M value satisfy the same tradeoff curve, i.e., with common values of the online time and tradeoff coefficient, adherence to M is guaranteed.

For the $\mathtt{p}\bar{\mathtt{D}}$ tradeoff, the similar work can be done to set $m_{\mathtt{p}\bar{\mathtt{D}}}$, $t_{\mathtt{p}\bar{\mathtt{D}}}$, and $\ell_{\mathtt{p}\bar{\mathtt{D}}}$ to achieve a desired probability of success $\mathtt{p}\bar{\mathtt{D}}_{\text{ps}}$, when a specific $\mathtt{p}\bar{\mathtt{D}}_{\text{msc}}$ value, together with any T and M that satisfy the tradeoff curve of Theorem 4.1 is given. We omit the detailed formulas.

The pc-tc curves for the original DP, parallel DP, perfect DP, and perfect parallel DP tradeoffs are given in Figure 5.1 for some specific success rates. As summarized at the ends of Section 2.3.1, Section 3.1.2, and Section 4.2, all the DP methods, mentioned in this work, allocate slightly more than $\log m$ bits to store each table entry. Since all the DP methods choose m value at a common level, together with t and ℓ in a typical situation, comparisons of $\mathtt{D}_{TM^2}$, $\mathtt{pD}_{TM^2}$, $\bar{\mathtt{D}}_{TM^2}$, and $\mathtt{p}\bar{\mathtt{D}}_{TM^2}$ will be fair under consideration of the adjustment of tradeoff coefficients, explained in the previous section. Within each box, being lower corresponds to having better online efficiency and being closer to the left edge corresponds to requiring less pre-computation. Hence, one may roughly interpret being situated closer to the lower left corner as displaying better performance. In each framed

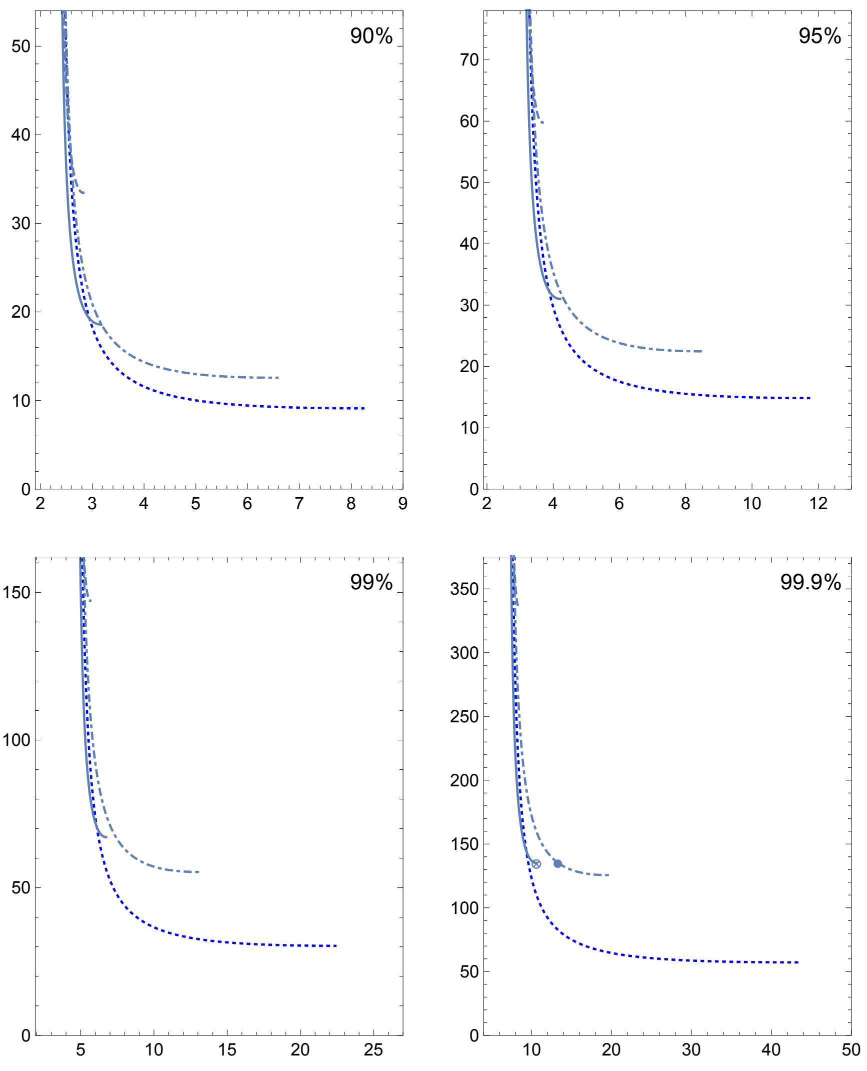

Figure 5.1: ($\mathtt{D}_{\text{pc}}$, $\mathtt{D}_{TM^2}$)(dashed), ($\mathtt{D}_{\text{pc}}$, $\mathtt{pD}_{TM^2}$)(line), ($\bar{\mathtt{D}}_{\text{pc}}$, $\bar{\mathtt{D}}_{TM^2}$)(dotdashed), and ($\bar{\mathtt{D}}_{\text{pc}}$, $\mathtt{p}\bar{\mathtt{D}}_{TM^2}$)(dotted), at various success rates.

graph box, all the curves should be seen as extending infinitely upwards. However, the right ends of the graphs are clearly visible. The curves start to go back up beyond these right ends, so that these right ends correspond to the minimum tradeoff coefficient achievable by each algorithms. As going beyond this minimum implies using larger pre-computation to obtain worse tradeoff efficiency, so parameters corresponding to the parts that are not drawn should not be used.

In each box, the dashed line, blue line, dotdashed line, and dotted line represent the choice of $(\mathtt{D}_{\mathrm{pc}}, \mathtt{D}_{TM^2})$, $(\mathtt{D}_{\mathrm{pc}}, \mathtt{pD}_{TM^2})$, $(\bar{\mathtt{D}}_{\mathrm{pc}}, \bar{\mathtt{D}}_{TM^2})$, and $(\bar{\mathtt{D}}_{\mathrm{pc}}, \mathtt{p}\bar{\mathtt{D}}_{TM^2})$ pairs made available by the original, parallel, perfect, and perfect parallel DP tradoe-offs, respectively. Note that, it is reasonable to treat the parameter $\mathtt{X}_{\mathrm{msc}} = \frac{mt^2}{\mathsf{N}}$ that was used to draw these graphs as a continuous variable, even though it originates from integers.

It is quite clear from Figure 5.1 that the perfect parallel DP tradeoff (dotted line) is more likely to be preferable over the other DP algorithms. Even though the $\mathtt{pD}$ tradeoff has a small advantage of low pre-computation cost against the perfect DP variants, the $\mathtt{p}\bar{\mathtt{D}}$ tradeoff has a much bigger advantage on online performance by comparison.

When the performances of the $\mathtt{pD}$ and $\bar{\mathtt{D}}$ tradeoffs are compared against each other, one can conclude that the $\mathtt{pD}$ tradeoff (line) could be less useful than the perfect DP tradeoff (dotdashed line), at 90%, 95%, and 99% success requirements, unless one is extremely constrained in the amount of pre-computation possible. On the other hand, as seen in the last graph box in Figure 5.1, $\otimes$-option of the $\mathtt{pD}$ tradeoff is more likely to be chosen, compared with the right end option of the perfect DP tradeoff. Because, $\otimes$-option allow us to achieve similar tradeoff efficiency of the right end option of the perfect DP tradeoff at a visibly lower pre-computation cost. In addition, $\bullet$-option of the perfect DP tradeoff, which provides the same online efficiency with $\otimes$-option of the $\mathtt{pD}$ tradeoff, is hardly chosen.

Let us explain the reason why the $\mathtt{pD}$ tradeoff is getting more efficient when the higher success rate is required in comparison against the perfect DP tradeoff. An ultimate purpose of parallel treatment on pre-computation tables is to spend more time in dealing with short online chains, so that the number of false alarms is expected to be reduced. However, at a non-high success requirement, it is meaningless, since almost all online chains will be treated until the correct answer **p** is found. On the other hand, if extremely high success rate is required, the efficiency of the $\mathtt{pD}$ tradeoff increases much more, because the algorithm terminates before all the online chains reach its DP so all possible alarms are resolved, so that alarms produced by short online chains are treated firstly. For the same reason, as seen in Figure 5.2, two graphs of the perfect DP and $\mathtt{p\bar{D}}$ tradeoffs are very close to each other, while the $\mathtt{pD}$ tradeoff has bad performance at 25% success rate.

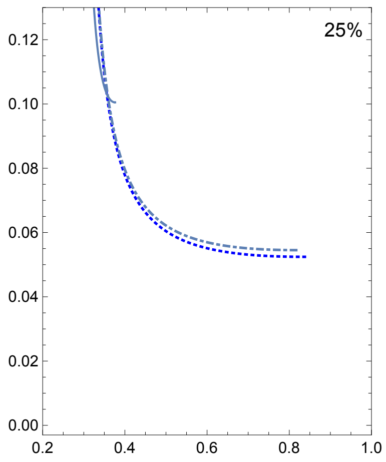

Figure 5.2: ($\mathtt{D}_{\mathrm{pc}}$, $\mathtt{pD}_{TM^2}$)(line), ($\mathtt{\bar{D}}_{\mathrm{pc}}$, $\mathtt{\bar{D}}_{TM^2}$)(dotdashed), and ($\mathtt{\bar{D}}_{\mathrm{pc}}$, $\mathtt{p\bar{D}}_{TM^2}$)(dotted), at 25% success rate.

In conclusion, both use of perfect tables and parallel treatment of pre-computation tables have positive effects, compared to the performance of the original DP tradeoff. Also, one can safely conclude that the perfect parallel DP tradeoff outperforms the other DP tradeoff algorithms for every situation.

5.3 $\mathtt{p\bar{D}}$ vs. Rainbow

Now we are going to compare the $\mathtt{p\bar{D}}$ tradeoff, the most efficient tradeoff method between the mentioned four DP algorithms, to the non-perfect $\mathtt{pR}$ and perfect $\mathtt{p\bar{R}}$ tradeoffs. As before, comparisons will be made at a few common fixed probabilities of success, so the symbols $\bar{\mathrm{pD}}_{\mathrm{ps}}$, $\mathrm{pR}_{\mathrm{ps}}$, and $\bar{\mathrm{pR}}_{\mathrm{ps}}$ will now be treated as fixed constants.

Let us explain how the pc-tc curve for the perfect rainbow tradeof may be plotted. One can combine (3.17) and Proposition 3.6 to express the pre-computation coefficient

$$\bar{\mathrm{pR}}_{\mathrm{pc}} = \frac{2}{2-\bar{\mathrm{pR}}_{\mathrm{msc}}}\big\{-\ln(1-\bar{\mathrm{pR}}_{\mathrm{ps}})\big\} = -\frac{2\ell\ln(1-\bar{\mathrm{pR}}_{\mathrm{ps}})}{2\ell+\ln(1-\bar{\mathrm{pR}}_{\mathrm{ps}})} \tag{5.4}$$

as a function of the single variable ℓ, when $\bar{\mathrm{pR}}_{\mathrm{ps}}$ is treated as a fixed constant. Similarly, the substitution of (3.17) into the formula of Theorem 3.2 results in an expression for $\bar{\mathrm{pR}}_{TM^2}$ that is given in terms of the single variable ℓ. Thus, the pc-tc curve for the perfect rainbow tradeoff may be drawn as a curve parameterized by ℓ. As with the DP tradeoff, the possibility of tradeoffs between online time and storage requirement remains unaffected by the restrictions on $\bar{\mathrm{pR}}_{\mathrm{ps}}$ and ℓ.

The pc-tc curve for the non-perfect rainbow tradeoff can be plotted in the similar way to the perfect rainbow tradeoff. One can combine (2.6) and (2.7) to express the pre-computation coefficient

$$\mathrm{pR}_{\mathrm{pc}} = 2\ell\left((1-\mathrm{pR}_{\mathrm{ps}})^{-\frac{1}{2\ell}}-1\right)$$

as a function of the single variable ℓ, when $\mathrm{pR}_{\mathrm{ps}}$ is treated as a fixed constant. Similarly, from (2.7), one can express $\mathrm{pR}_{\mathrm{msc}}$ as a function of ℓ and $\mathrm{pR}_{\mathrm{ps}}$. Substituting it into the formula (2.8), pR_{TM^2} can be also regarded as a function of the single variable ℓ, at a fixed success probability. Now, the pc-tc curve for the non-perfect rainbow tradeoff may be drawn as a curve parameterized by ℓ as like the perfect rainbow tradeoff.

Before plotting the pc-tc curves and comparing each other, let us review the adjusted tradeoff coefficients, explained in Section 5.1. Due to the relatively differences in the allocated number of bits to store a single table entry by each tradeoff algorithm after applying the storage optimization techniques, have been analyzed for each tradoff method, the comparison of online efficiencies must be made between adjusted tradeoff coefficients.

Although no single set of adjustment factors can be appropriate for all situations, we still need to fix them to something specific in order to proceed with the comparison in this work. Our choice, which will soon be justified, is to compare the adjusted tradeoff coefficients $\frac{1}{4}\bar{\mathtt{pD}}_{TM^2}$, $\mathtt{pR}_{TM^2}$, and $\bar{\mathtt{pR}}_{TM^2}$ against each other. At the ends of Section 2.3.2, Section 4.2 and Section 3.2.2, we had stated that all of the $\bar{\mathtt{pD}}$, non-perfect, and perfect rainbow tradeoffs need to allocate "slightly more than $\log m$ bits" to record each table entry. Now, for the $\bar{\mathtt{pD}}$ tradeoffs, the total complexity, defined as the sum $T+M$, is minimized by the parameters $m \approx t \approx \ell \approx \mathsf{N}^{\frac{1}{3}}$, and the same for the perfect and non-perfect rainbow tradeoffs is minimized by the parameters $m \approx \mathsf{N}^{\frac{2}{3}}$, $t \approx \mathsf{N}^{\frac{1}{3}}$, and a small ℓ. In fact, we could state that these same parameters are used in practical implementations, as long as the approximations are understood to be extremely crude. Hence, the "slightly more than $\log m$ bits" would often be not too far from $\frac{1}{3}\log\mathsf{N}$ bits and $\frac{2}{3}\log\mathsf{N}$ bits for the $\bar{\mathtt{pD}}$ tradeoff and two rainbow tradeoffs, respectively. In this sense, the $\bar{\mathtt{pD}}$ tradeoff require only half as many bits as the two rainbow tradeoffs in storing each table entry, and our choice of the adjusted tradeoff coefficients is somewhat justified.

The pc-tc curves for the $\bar{\mathtt{pD}}$, non-perfect and perfect rainbow tradeoffs are given in Figure 5.3 for some specific success rates. As explained in the previous section, within each box, being lower corresponds to having better online efficiency and being closer to the left edge corresponds to requiring less precomputation. Hence, one may roughly interpret being situated closer to the lower left corner as displaying better performance.

In each box, the empty circles represent the choice of $(\bar{\mathtt{pR}}_{\mathrm{pc}}, \bar{\mathtt{pR}}_{TM^2})$-pairs made available by the perfect rainbow tradeoff, and the filled dots represent data for the non-perfect rainbow tradeoff. Each circle for the perfect rainbow tradeoff corresponds to an integer ℓ value, with the rightmost circle of each box corresponding to $\ell = \lceil -\frac{1}{2}\ln(1-\bar{\mathtt{pR}}_{\mathrm{ps}})\rceil$, as determined by the bound (3.18). Since the table count ℓ must be an integer, the available choices appear as a discrete set of circles. Similar statements may be made for the dots that represent data for the non-perfect rainbow tradeoff. The rightmost dot of each box, which represents the smallest tradeoff coefficient of the non-perfect tradeoff, was determined in the previous work [18]. The line in each box represents data for the $\bar{\mathtt{pD}}$ tradeoffs. As mentioned before, unless N is small, it is reasonable to treat the parameter $\bar{\mathtt{pD}}_{\text{msc}} = \frac{mt^2}{\mathsf{N}}$ that was used to draw these graphs as a continuous variable, even though it originates from integers. One can numerically verify that the tradeoff coefficient $\bar{\mathtt{pD}}_{TM^2}$ attains its minimum at $\bar{\mathtt{pD}}_{\text{msc}} = 2.649$, 2.929, 3.662, and 4.683 for 90%, 95%, 99%, and 99.9% probabilities of success, respectively.

It is quite clear from Figure 5.3 that the $\bar{\mathtt{pD}}$ tradeoff (line) is less desirable than the perfect rainbow tradeoff (circle), regardless of how one wants to balance online efficiency against pre-computation cost. It also seems fair to claim that the perfect rainbow tradeoff (circle) is at an advantage over the non-perfect rainbow tradeoff (dot), since it can approximately provide every option made available by the non-perfect rainbow, while providing many more options that cannot be approximated by the non-perfect rainbow tradeoff. Furthermore, the perfect rainbow tradeoff presents the possibility of obtaining much better online efficiencies, although these must be paid for with higher pre-computation costs.

One may say that the $\bar{\mathtt{pD}}$ tradeoff (line) could be more useful than the perfect rainbow tradeoff (circle) by flexiblities in choosing options at the 90% success rate. For example, the perfect rainbow tradeoff cannot provide the option corresponding to the $\swarrow$-option of the $\bar{\mathtt{pD}}$ tradeoff. However, the circle next to the rightmost circle of the perfect rainbow tradeoff is more attractive option against the $\swarrow$-option $(\bar{\mathtt{D}}_{\text{pc}}, \bar{\mathtt{pD}}_{TM^2}) = (4.81674,\ 2.54876)$, because that option allows us to obtain slightly lower, but very similar online efficiency, $\bar{\mathtt{pR}}_{TM^2} = 2.68737$ with investment in much smaller pre-computation $\bar{\mathtt{pR}}_{\text{pc}} = 3.73653$. Thus, still, the perfect rainbow tradeoff outperforms the $\bar{\mathtt{pD}}$ tradeoff.

In all situations, one can conclude that the $\bar{\mathtt{pD}}$ tradeoff (line) is more preferable than the non-perfect rainbow tradeoff (dots), unless one is extremely constrained in the amount of pre-computation possible.

The success rates covered by Figure 5.3 are those that would be of practical interest. However, we acknowledge that for success rate requirements that are much lower, such as 25% or 50%, the situation is somewhat different. One can easily verify through curves similar to those of Figure 5.4 that the use of the $\mathtt{p\bar{D}}$ tradeoff can be advisable at these less interesting low success rates.

The comments we have given so far concerning Figure 5.3 should generally be acceptable, but when lowering the pre-computation cost is immensely important, there remains a small possibility that the non-perfect rainbow tradeoff (dot) could be preferred over the perfect rainbow tradeoff (circle). This is illustrated by Figure 5.5, which is an enlarged view of a small rectangular part from the 99% box of Figure 5.3. We have intentionally stretched the small rectangle in the horizontal direction and have reduced the height, so that even a small difference in the pre-computation coefficient is perceived as being significant. Even though some sacrifice in the online efficiency is inevitable, the options provided by the non-perfect rainbow tradeoff (dot) now seem much more reasonable than previously felt when viewed from within Figure 5.3.

To summarize, when the online efficiency and pre-computation cost are both taken into account, the perfect rainbow tradeoff is very likely to be advantageous over $\mathtt{p\bar{D}}$, non-perfect rainbow tradeoffs, in typical situations. However, there may be special circumstances under which the preferences could be different. For example, importance of lowering the pre-computation cost may shift the preference towards the non-perfect rainbow tradeoff, and the need for fine-tuned parameter choices may make the $\mathtt{p\bar{D}}$ tradeoff favorable at low success rate requirements.

Before ending this section, let us add remark concerning the range in which (4.8) is accurate. As in Section 4.3, the empirical result (4.8) is reliable at least for parameter choices satisfying $0 < \mathtt{p\bar{D}}_{\mathrm{msc}} \leq 2.4$. The options given by $\mathtt{p\bar{D}}_{\mathrm{msc}} = 2.4$ are indicated as bold dots $\bullet$ in all graph boxes of Figure 5.3. Under empirical experiences, our formula (4.8) tends to be getting higher than practical test values obtained by experiments at the bigger $\mathtt{p}\bar{\mathtt{D}}_{\mathrm{msc}}$ than 2.4, so we can expect that the curves after the • point would not be lower than those in Figure 5.3, at least. Thus, one may safely claim that options over the option given by $\mathtt{p}\bar{\mathtt{D}}_{\mathrm{msc}} = 2.4$ are meaningless in a certain sense that to achieve a little better online efficiency by invest much more pre-computation resources is much less preferable. Therefore, our conclusion of this section is trustworthy, despite of unsureness of the empirical formula (4.8) over $\mathtt{p}\bar{\mathtt{D}}_{\mathrm{msc}} = 2.4$.

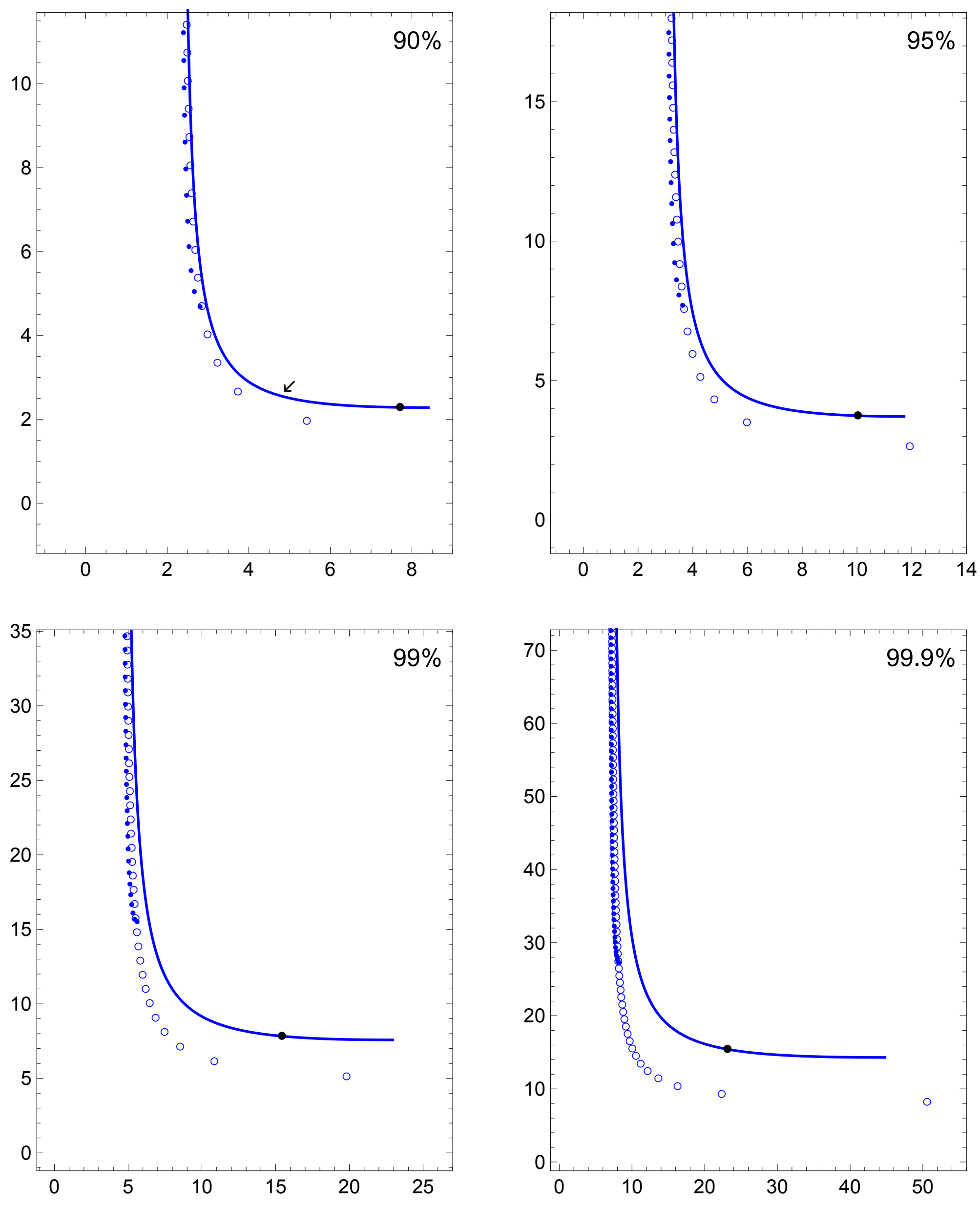

Figure 5.3: $(\bar{\mathtt{D}}_{\mathrm{pc}}, \frac{1}{4}\mathtt{p}\bar{\mathtt{D}}_{TM^2})$(line), $(\mathtt{pR}_{\mathrm{pc}}, \mathtt{pR}_{TM^2})$(dots), and $(\mathtt{p}\bar{\mathtt{R}}_{\mathrm{pc}}, \mathtt{p}\bar{\mathtt{R}}_{TM^2})$(empty circles), at high success rates.

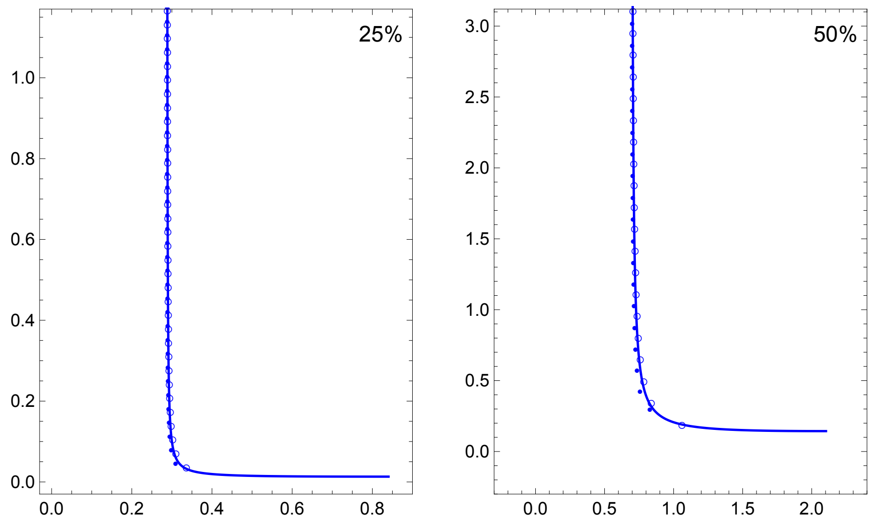

Figure 5.4: ($\bar{\mathtt{D}}_{\mathrm{pc}}$, $\frac{1}{4}\mathtt{p}\bar{\mathtt{D}}_{TM^2}$)(line), ($\mathtt{pR}_{\mathrm{pc}}$, $\mathtt{pR}_{TM^2}$)(dots), and ($\mathtt{p}\bar{\mathtt{R}}_{\mathrm{pc}}$, $\mathtt{p}\bar{\mathtt{R}}_{TM^2}$)(empty circles), at low success rates.

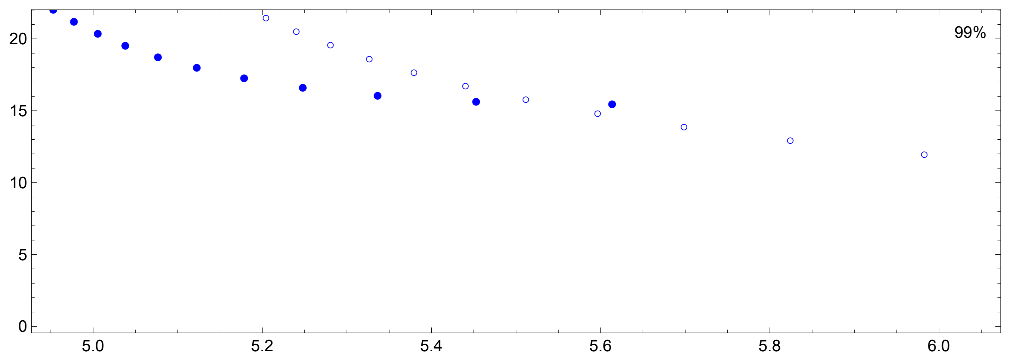

Figure 5.5: Tradeoff coefficient for perfect (circles) and non-perfect (dots) rainbow tradeoffs in relation to their pre-computation costs at 99% success rate.

Chapter 6

Practice-Oriented Comparison

In this chapter, practical performance abilities of the perfect parallel DP $\bar{\mathtt{pD}}$ and perfect rainbow $\bar{\mathtt{pR}}$ tradeoffs will be compared against the practically used perfect rainbow algorithm, namely, $\bar{\mathtt{sR}}$ tradeoff. A theoretical comparison of these algorithms, following the approach explained in Section 5.1, takes the following aspects of the tradeoff algorithms into account: (a) the expected computational complexity of the online phase that includes the effects of false alarms, (b) the size of physical storage space required to retain the pre-computation tables, (c) computational complexity of the pre-computation phase. Even though such a comparison could be meaningful in many situations, we have learned that practioners treated such a comparison as being theoretical.

At least three reasons can be stated as to why the approach is seen as inadequate. First, the factor that affects the user experience most directly is the speed of inversion, but the online computational complexity does not accurately reflect the physical time taken by the online phase. The inaccuracy is mainly due to the additional time taken to access the pre-computation tables, which the approaches of previous chapters and [18] had completely ignored. Second, there is a practical limit to how much storage can be utilized, at which point any measure of tradeoff characteristic looses its meaning. The monetary cost of data storage devices has become so low that some might even claim that the effort of securing storage space up to a few terabytes almost does not depend on the size. On the other hand, to utilize anything much larger, one must either greatly sacrifice disk access speed or invest much more on the connection between the online system and the storage device. This discontinuity implies that the theoretical tradeoff coefficient is of limited value in representing the capabilities of a tradeoff algorithm. Third, in most cases, the complexity of the pre-computation phase is not very important. The cost of pre-computation can become critical when the cost approaches the maximum amount of available resources. However, in practice, since one can initially provide an inversion service of low success rate and later increase the success rate by adding more pre-computation tables as they become available, the upper bound to resources is a very flexible concept. Furthermore, since choosing the search spaces in a straightforward manner leads to search spaces with quantum differences between their sizes rather than those that form a continuous spectrum of sizes, the pre-computation task for each search space choice is likely to be either quite infeasible or easily possible, rather than right at the boundary of feasibility.

These observations show that, to be meaningful in practice, a comparison of tradoff algorithms must be focused on the physical (wall-clock) time requirements of the online phases. Such a comparison of the $\mathtt{s\bar{R}}$, $\mathtt{p\bar{R}}$, and $\mathtt{p\bar{D}}$ tradeoffs will be provided in this chapter.

From now on, we refer $T^{\mathtt{wc}}_{\mathtt{X}}$ as the total physical wall-clock time requirement of the online phase for each tradeoff algorithm $\mathtt{X}$. The readers should distinguish the notion of $T^{\mathtt{wc}}$ with T, which has been used to refer the expected number of one-way function invocations required to perform the online phase of tradeoff algorithm. In order to derive the formulas of the physical times $T^{\mathtt{wc}}_{\mathtt{p\bar{D}}}$ and $T^{\mathtt{wc}}_{\mathtt{p\bar{R}}}$ explicitly, additional costs for the $\mathtt{p\bar{R}}$, and $\mathtt{p\bar{D}}$ tradeoffs should be computed, and those will be given in the next section. And then, analysis of the $\mathtt{s\bar{R}}$ tradeoff will be following. Most of the arguments given there are reformulations of similar arguments appearing in the recent work [21].

6.1 Additional Costs for the $\mathtt{p\bar{D}}$ and $\mathtt{p\bar{R}}$ Tradeoffs

In this section, the expected numbers of table lookups to be required by the online phases of the $\mathtt{p\bar{D}}$ and the $\mathtt{p\bar{R}}$ tradeoffs are presented. We use a symbol H to represent the expected number of online phase table searches, and it was chosen so that the words <u>H</u>ash table could be associated with it. All of notations and terminologies will be also used as before.

Proposition 6.1. *The online phase of the* $\mathtt{p\bar{D}}$ *tradeoff is expected to generate*

$$H_{\mathtt{p\bar{D}}} = t \times \frac{\mathtt{p\bar{D}}_{\mathrm{ln}}}{\mathtt{p\bar{D}}_{\mathrm{msc}}(\mathtt{p\bar{D}}_{\mathrm{ln}}-1)}\left\{1-\left(1+\frac{\mathtt{p\bar{D}}_{\mathrm{msc}}}{2}\right)^{1-\mathtt{p\bar{D}}_{\mathrm{ln}}}\right\}$$

searches to the perfect DP tables.

Proof. Since a single search is performed precisely when a DP is reached during the generation of the online chains, and since, on average, one in every t iterations of the one-way function produces a DP, the expected number of table lookups is $\frac{1}{t}$ of the number of one-way function iterations given by Proposition 4.1. $\square$

Now, let us introduce the memory hash table lookups tradeoff curve $MH \approx \mathsf{N}$, referred as the *M*-*H* tradeoff curve. Terminologies $\mathtt{p\bar{D}}_{MH}$ and $\mathtt{p\bar{R}}_{MH}$ will be used to denote the *M*-*H* tradeoff coefficients $\frac{MH}{\mathsf{N}}$ of the $\mathtt{p\bar{D}}$ and $\mathtt{p\bar{R}}$ tradeoffs, respectively.

The following theorem is direct consequences of Proposition 6.1 and the fact that the pre-computation tables contain $M = m\ell$ entries in total.

Theorem 6.1. *Let the symbol H denote the expected number of online phase table searches for the* $\mathtt{p\bar{D}}$ *tradeoff. Then* $\frac{MH}{\mathsf{N}}$ *equals*

$$\mathtt{p\bar{D}}_{MH} = \frac{\mathtt{p\bar{D}}_{\mathrm{ln}}^{2}}{2(\mathtt{p\bar{D}}_{\mathrm{ln}}-1)}\left\{1-\left(1+\frac{\mathtt{p\bar{D}}_{\mathrm{msc}}}{2}\right)^{1-\mathtt{p\bar{D}}_{\mathrm{ln}}}\right\},$$

where $\mathtt{p\bar{D}}_{\mathrm{ln}} = -\frac{\ln(1-\mathtt{p\bar{D}}_{\mathrm{ps}})}{\ln(1+\frac{\mathtt{p\bar{D}}_{\mathrm{msc}}}{2})}$.

As with the time memory tradeoff curve in Theorem 4.1, $\bar{\mathtt{pD}}_{MH}$ remains constant under any fixed pair of $\bar{\mathtt{pD}}_{\mathrm{msc}}$ and $\bar{\mathtt{pD}}_{\mathrm{ps}}$ values. Hence, the above theorem states the possibility of utilizing T-M and M-H tradeoffs.

Similar claims for the $\bar{\mathtt{pR}}$ tradeoff can be obtained as below.

Proposition 6.2. *The online phase of the* $\bar{\mathtt{pR}}$ *tradeoff is expected to generate*

$$H_{\bar{\mathtt{pR}}} = t \times \frac{\bar{\mathtt{pR}}_{\mathrm{ps}}}{\bar{\mathtt{pR}}_{\mathrm{msc}}}$$

searches to the perfect rainbow tables.

Proof. The number of table lookups for the $\bar{\mathtt{pR}}$ tradeoff can be states as

$$H_{\bar{\mathtt{pR}}} = \ell \sum_{i=1}^{t} \Big(1 - \frac{m}{\mathsf{N}}\Big)^{\ell(i-1)} = \ell \frac{1-(1-\frac{m}{\mathsf{N}})^{\ell t}}{1-(1-\frac{m}{\mathsf{N}})^{\ell}} = \frac{\bar{\mathtt{pR}}_{\mathrm{ps}}\ell}{1-(1-\frac{m\ell}{\mathsf{N}}+O(\frac{1}{t}))} = \frac{\bar{\mathtt{pR}}_{\mathrm{ps}}}{\bar{\mathtt{pR}}_{\mathrm{msc}}} t$$

where the last equality ignores a multiplicative factor of $1 + O\big(\frac{1}{t}\big)$. □

Theorem 6.2. *Let the symbol H denote the expected number of online phase table searches for the* $\bar{\mathtt{pR}}$ *tradeoff. Then $\frac{MH}{\mathsf{N}}$ equals*

$$\bar{\mathtt{pR}}_{MH} = \bar{\mathtt{pR}}_{\mathrm{ps}}\ell.$$

6.2 Analysis of the $\bar{\mathtt{sR}}$ Tradeoff

In this section, we prepare information concerning the $\bar{\mathtt{sR}}$ tradeoff that is analogous to Theorem 3.2 and Theorem 6.2. As in the previous sections, the symbols T and M will denote the expected numbers of online phase one-way function computations, and pre-computation table entries, respectively.

Recall that the $\bar{\mathtt{sR}}$ tradeoff compares multiple online chain ending points against the ending points of the pre-computation tables. Let us write L or $L_{\bar{\mathtt{sR}}}$ to denote the expected number of pre-computation tables entries that would be compared against the online chain ending points. The character L is used because each of these table entries must be Loaded into memory for comparison.

The work [21] expressed the complexities $T_{\mathtt{s\bar{R}}}$ and $L_{\mathtt{s\bar{R}}}$ for the $\mathtt{s\bar{R}}$ tradeoff as functions of the algorithm parameters. It is not too difficult to derive the following results from their formulas.

Theorem 6.3. *The T-M tradeoff curve for the* $\mathtt{s\bar{R}}$ *tradeoff are* $TM^2 = \mathtt{s\bar{R}}_{TM^2}\mathsf{N}^2$, *where the tradeoff coefficient is given by*

$$\mathtt{s\bar{R}}_{TM^2} = \mathtt{s\bar{R}}_{\mathrm{ps}}\{\ln(1-\mathtt{s\bar{R}}_{\mathrm{ps}})\}^2 \left\{ \frac{1}{2\bigl(1-(1-\mathtt{s\bar{R}}_{\mathrm{ps}})^{\frac{1}{\ell}}\bigr)} + \frac{1}{6} + \frac{\ln(1-\mathtt{s\bar{R}}_{\mathrm{ps}})}{48\ell} \right\}.$$

Theorem 6.4. *Let the symbol L denote the expected number of pre-computation table entries loaded into memory during the online phase of the* $\mathtt{s\bar{R}}$ *tradeoff. Then* $\frac{L}{M}$ *equals*

$$\mathtt{s\bar{R}}_{LM^{-1}} = -\frac{\mathtt{s\bar{R}}_{\mathrm{ps}}}{\ln(1-\mathtt{s\bar{R}}_{\mathrm{ps}})}.$$

6.3 Expressions for the Physical Online Time

We must start by introducing three system constants τ_F, τ_L, and τ_H.

Let τ_F be the physical time, for example, in milliseconds, taken by a single iteration of the colored one-way function, i.e., the composition of the one-way function and the reduction function, on the online phase platform. This value will depend on the targeted one-way function, the reduction function being used, the capabilities of the online phase platform, and the implementation itself.

Recall that the $\mathtt{s\bar{R}}$ tradeoff checks whether each ending point from the pre-computation table matches any one of the t online chain ending points generated in one batch. In practice, each of the pre-computation table entries must be copied from slow disk into fast system memory before being processed, and the time taken for these data loading operations greatly overwhelms the time taken for the comparison itself. In fact, the possibility of loading the tables faster is an incentive to reducing storage size that is more meaningful in practice than the reduction in storage cost. Let us write τ_L to denote the physical time taken to load one table entry from slow disk to fast system memory. This will depend on the sequential disk read speed of the online phase platform and on how many bytes are allocated to each pre-computation table entry. Current implementations apply the index file technique to pre-computation tables stored on disk and expand each table entry into its original full form during memory loading. Since the expansion requires very little computational effort, the compactification of data stored on disk has the effect of reducing loading time.

The final system constant we introduce is the physical time τ_H taken by a single search of the pre-computation table for a matching ending point. Our interest lies in the case where the pre-computation tables are too large to fit within the fast system memory and accesses to the slow disk storage are inevitable to perform table lookups. Note that if the search times for the successful and failed ending point matches are significantly different, one must compute a weighted average value that takes the probability for the online chain to merge into the pre-computation matrix into account. The constant τ_H will certainly be affected by the random disk read speed of the online phase machine and will depend greatly on the implementation.

Using the notation we have introduced, the expected physical times taken by the online phases of the three tradeoffs can be written as follows.

$$T^{\mathtt{wc}}_{\mathtt{s\bar{R}}} = \tau_F T_{\mathtt{s\bar{R}}} + \tau_L L_{\mathtt{s\bar{R}}}.$$
$$T^{\mathtt{wc}}_{\mathtt{p\bar{R}}} = \tau_F T_{\mathtt{p\bar{R}}} + \tau_H H_{\mathtt{p\bar{R}}}.$$
$$T^{\mathtt{wc}}_{\mathtt{p\bar{D}}} = \tau_F T_{\mathtt{p\bar{D}}} + \tau_H H_{\mathtt{p\bar{D}}}.$$

Substituting Theorem 6.3, Theorem 6.4, Theorem 3.2, Theorem 6.2, Theorem 4.1, and Theorem 6.1 into these equations, we can derive the expressions

$$\frac{1}{\tau_F} T^{\mathtt{wc}}_{\bar{\mathtt{sR}}} = \bar{\mathtt{sR}}_{TM^2}\Big(\frac{\mathsf{N}}{M}\Big)^2 + \bar{\mathtt{sR}}_{LM^{-1}}\frac{\tau_L}{\tau_F} M, \tag{6.1}$$

$$\frac{1}{\tau_F} T^{\mathtt{wc}}_{\bar{\mathtt{pR}}} = \bar{\mathtt{pR}}_{TM^2}\Big(\frac{\mathsf{N}}{M}\Big)^2 + \bar{\mathtt{pR}}_{MH}\frac{\tau_H}{\tau_F}\Big(\frac{\mathsf{N}}{M}\Big), \tag{6.2}$$

$$\frac{1}{\tau_F} T^{\mathtt{wc}}_{\bar{\mathtt{pD}}} = \bar{\mathtt{pD}}_{TM^2}\Big(\frac{\mathsf{N}}{M}\Big)^2 + \bar{\mathtt{pD}}_{MH}\frac{\tau_H}{\tau_F}\Big(\frac{\mathsf{N}}{M}\Big). \tag{6.3}$$

6.4 How to Minimize the Physical Online Time

Let us discuss how one might minimize the the online times given by (6.1), (6.2), and (6.3). For the remainder of this chapter, we assume the three tradeoff algorithms are set to achieve a fixed success rate requirement $\bar{\mathtt{sR}}_{\mathrm{ps}} = \bar{\mathtt{pR}}_{\mathrm{ps}} = \bar{\mathtt{pD}}_{\mathrm{ps}}$ that is common to the three algorithms. In fact, any reasonable comparison of tradeoff algorithms would require a common success rate to the algorithms.

After drawing a few graphs, one can be confident that $\bar{\mathtt{sR}}_{TM^2}$, as given by Theorem 6.3, is an increasing function of ℓ, for any fixed value of $\bar{\mathtt{sR}}_{\mathrm{ps}}$. Also note that $\bar{\mathtt{sR}}_{LM^{-1}}$, as given by Theorem 6.4, is constant for each choice of $\bar{\mathtt{sR}}_{\mathrm{ps}}$. Hence, to minimize the righthand side of (6.1), one must use the smallest possible value for ℓ, which is

$$\ell = \Big\lceil -\frac{1}{2}\ln(1-\bar{\mathtt{sR}}_{\mathrm{ps}})\Big\rceil. \tag{6.4}$$

Once the values of $\bar{\mathtt{sR}}_{TM^2}$ and $\bar{\mathtt{sR}}_{LM^{-1}}$ in (6.1) have been fixed through the above choice of ℓ, the righthand side function of M can be minimized by setting

$$M = \Big(2\frac{\bar{\mathtt{sR}}_{TM^2}}{\bar{\mathtt{sR}}_{LM^{-1}}}\frac{\tau_F}{\tau_L}\Big)^{\frac{1}{3}}\mathsf{N}^{\frac{2}{3}}. \tag{6.5}$$

One can similarly check that both $\bar{\mathtt{pR}}_{TM^2}$ and $\bar{\mathtt{pR}}_{MH}$ of (6.2), as given by the formulas of Theorem 3.2 and Theorem 6.2, are increasing functions of ℓ, for each fixed value of $\bar{\mathtt{pR}}_{\mathrm{ps}}$. Hence, as with (6.1), given a success rate requirement $\bar{\mathtt{pR}}_{\mathrm{ps}}$, one must use

$$\ell = \left\lceil -\frac{1}{2}\ln(1-\bar{\mathtt{pR}}_{\mathrm{ps}})\right\rceil, \tag{6.6}$$

in order to minimize the righthand side of (6.2), for any fixed value of M. However, unlike the (6.1) case, the righthand side of (6.2) approaches zero as M is increased. This is not surprising, since the limit corresponds to the dictionary attack, which requires just a single search of the pre-computation table. Hence, one can minimize (6.2) by using the largest possible M, together with (6.6).

The situation with (6.3) is slightly more complicated. Even though both $\bar{\mathtt{pD}}_{TM^2}$ and $\bar{\mathtt{pD}}_{MH}$, as given by Theorem 4.1 and Theorem 6.1, are functions of the single variable $\bar{\mathtt{pD}}_{\mathrm{msc}}$, assuming a fixed success rate requirement, one can find that the two are minimized at different $\bar{\mathtt{pD}}_{\mathrm{msc}}$ values. Hence, to minimize (6.3), one must first fix M to the largest possible value and then find the optimal value of $\bar{\mathtt{pD}}_{\mathrm{msc}}$ for that M.

6.5 Comparisons

Certain numbers associated with the targeted inversion problem, available resources, and online platform characteristics need to be fixed before the comparisons of algorithm performances are provided. The discussion below will first be carried out with these numbers fixed to specific realistic values. This will later be followed by a table that concisely presents the algorithm performances when these numbers are replaced with other realistic values.

The search space size will be fixed to

$$\mathsf{N} = \sum_{i=1}^{8} 95^i = 2^{52.574},$$

which corresponds to the set of all passwords of lengths up to 8, constructed from all 95 characters on the standard keyboard. The success rate requirement will be set to

$$\bar{\mathtt{sR}}_{\mathrm{ps}} = \bar{\mathtt{pR}}_{\mathrm{ps}} = \bar{\mathtt{pD}}_{\mathrm{ps}} = 99.9\%,$$

which is the claimed success rate of the majority of tables created by the two rainbow table distributors [1,3]. The distributor [2] provides the tables that assure success rates less than 99.9%, except for NT-hashes of only numeric passwords. The search space size $\mathsf{N} = 2^{52.574}$ is the largest among the search space dealt with by the distributor [1], and is even larger than the charged tables provided by the distributor [2]. The largest key space for the distributor [3] treats all passwords of lengths up to 9, constructed from all uppercase and lowercase alphabets and numeric characters, and its size is $2^{53.611}$, almost twice our fixed search space size N. However, since those tables assure the success rate 96.8%, the time for creation of those rainbow tables will be $3.40 \times 10^{17} \times \tau_F$ from (5.4). This is of similar level with ours, which requires

$$\bar{\mathtt{sR}}_{\mathrm{pc}}\mathsf{N} = \frac{-2\ell \ln(1-\bar{\mathtt{sR}}_{\mathrm{ps}})}{2\ell + \ln(1-\bar{\mathtt{sR}}_{\mathrm{ps}})}\mathsf{N} = \frac{-2\cdot 4\cdot \ln(1-0.999)}{2\cdot 4+\ln(1-0.999)}\mathsf{N} = 50.59\times\mathsf{N} = 3.39\times 10^{17}$$

invocations of one-way function for the pre-computations of tables. This facts suggest that pre-computation requirement for any larger search space is too large to be handled by entities other than well funded governmental organizations.

Currently, the cost of a 4 TB SATA-III 7200 RPM hard disk drive is less than \$250 and even 6 TB disks have recently become available. Solid state drives of 1 TB size are now available at approximately \$500. These SSDs may be eight times as expensive as the HDDs of the same size, but are still affordable for even personal use. A few of these HDDs or SSDs can easily be attached to a single PC, but to utilize a storage space of much larger size, one must sacrifice the speed of data access and work with a more elaborate hardware configuration. Based on these observations, we somewhat arbitrarily set the maximum storage size to 4 TB. It should be understood that this is not the bound on M, the number of table entries, which further depends on how many bytes are allocated to each table entry.

To fix the system constants, we made some measurements using specific implementations of one-way function, table loading operation, and table search operation. Details of these measurements will be explained in Appendix A.

Based on the measurements, we choose to use the realistic figure

$$\tau_F = 2.04 \times 10^{-7}\,\text{seconds}$$

as the physical time taken by a single iteration of the colored one-way function, when the targeted search space size is $\mathsf{N} = 2^{52.574}$. Assuming that 7 bytes are allocated to store a single table entry, we choose

$$\tau_L = 7.01 \times 10^{-8}\,\text{seconds}$$

as the physical time taken to load a single table entry from slow disk to fast system memory.

Let us now compute the optimal configurations for the $\bar{\mathtt{sR}}$ tradeoff. We learn from (6.4) that $\ell = 4$ must be used, and we can calculate the values $\bar{\mathtt{sR}}_{TM^2} = 35.2$ and $\bar{\mathtt{sR}}_{LM^{-1}} = 0.145$ from the formulas of Theorem 6.3. As stated by (6.5), the M that minimizes the online time is 3.99×10^{11}. Since $\log m = \log \frac{M}{\ell} = 36.5$, each starting point can be recorded well within 5 bytes through the use of sequential starting points, even though 37 bits may not be enough, due to the existence of discarded merging pre-computation chains. Recalling the ending point truncation technique, we know that it suffices to record only slightly more than 36.5 bits of each ending point. Note that a full set of 26-bit indices for the $\ell = 4$ tables consisting of 5-byte addresses require only 1.34 GB and can easily be held in the system memory of any modern PC. This shows that we can further remove 26 bits from each ending point and record the remaining slightly more than 10.5 bits within 2 bytes of space. Hence, the pre-computation tables occupy $(5+2) \times M$ bytes, which is 2.80 TB, of storage space and can fit comfortably within our 4 TB storage space bound. Finally, we can calculate (6.1) to state that the physical time of the online phase can be minimized to $T^{\mathtt{wc}}_{\bar{\mathtt{sR}}} = 2.98 \times 10^{10} \times \tau_F$.

The $\bar{\mathtt{pR}}$ tradeoff will be treated next. As before, we know from (6.6) that $\ell = 4$ tables are required, and we can compute from Theorem 3.2 and 6.2 that $\bar{\mathtt{pR}}_{TM^2} = 8.39$ and $\bar{\mathtt{pR}}_{MH} = 4.00$. Taking the $\bar{\mathtt{sR}}$ case as a hint, we assume each table entry can be recorded in 7 bytes and take $M = \frac{4\times 10^{12}}{7} = 5.71 \times 10^{11}$. Then, $\log m = \log \frac{M}{\ell} = 37.1$, so that $\log m_0 = 39.9$ from (3.14) and (3.17), to generate a perfect rainbow matrix which consists of m entries. Thus, assuming a 26-bit index to the sorted ending points, the 7 bytes assumption can be justified. Based on the measurements treating the pre-computation table, which consists of similar amount of table entries and which table entry is recorded in 7 bytes, we can choose to use the realistic figure

$$\tau_H^{\bar{\mathtt{pR}}} = 1.04 \times 10^{-2}\,\text{seconds}$$

The optimal physical online time can be calculated from (6.2) to be $T^{\mathtt{wc}}_{\bar{\mathtt{pR}}} = 3.55 \times 10^9 \times \tau_F$.

It remains to calculate the optimal online time for the $\bar{\mathtt{pD}}$ tradeoff. Noting that $\log \mathsf{N}^{\frac{1}{3}} = 17.5 < 3 \times 8$, we assume each table entry can be recorded in 5 bytes, and take $M = \frac{4\times 10^{12}}{5}$. Then, according to measurement from Appendix A working with random tables, targeted to perfect DP tables which entry was recorded in 5 bytes, we can set

$$\tau_H^{\bar{\mathtt{pD}}} = 0.939 \times 10^{-2}\,\text{seconds}.$$

The righthand side of (6.3) can be minimized to $T^{\mathtt{wc}}_{\bar{\mathtt{pD}}} = 5.25 \times 10^9 \times \tau_F$ at $\bar{\mathtt{pD}}_{\text{msc}} = 6.61$, using the formulas of Theorem 4.1 and Theorem 6.1. Recalling (4.2) and (4.3), we see that to achieve 99.9% success rate with $\bar{\mathtt{pD}}_{\text{msc}} = 6.61$, we must have $\frac{\ell}{t} = 2.37$, and this shows that $\log m = \log\left(\frac{M^2}{\bar{\mathtt{pD}}_{\text{msc}}\mathsf{N}} \frac{t^2}{\ell^2}\right) = 21.3$ must be used. We can also check that $\log t = 17.0$ must be used. Recalling (3.3), since $\log \vec{m}_0 = 23.4$, each starting point can be recorded in 3 bytes, as assumed before. As for the ending points, using the ending point truncation technique from Section 4.2,

Table 6.1: Optimal parameters and physical online time for $\mathsf{N} = 2^{52.574}$, 99.9% success rate, 4 TB storage, $\frac{\tau_L}{\tau_F} = 0.34$, $\frac{\tau_H^{\bar{\mathtt{pR}}}}{\tau_F} = 5.10 \times 10^4$, and $\frac{\tau_H^{\bar{\mathtt{pD}}}}{\tau_F} = 4.60 \times 10^4$.

alg.	$\log m$	$\log t$	ℓ	bytes/entry	online time	pre-comp
$\bar{\mathtt{sR}}$	36.5	16.8	4	7	$2.98 \times 10^{10} \times \tau_F$	$50.59\mathsf{N} \times \tau_F$
$\bar{\mathtt{pR}}$	37.1	16.3	4	7	$3.55 \times 10^{9} \times \tau_F$	$50.59\mathsf{N} \times \tau_F$
$\bar{\mathtt{pD}}$	21.3	17.0	$2.37 \times t$	5	$5.25 \times 10^{9} \times \tau_F$	$67.36\mathsf{N} \times \tau_F$

it suffices to record slightly more than 21.3 bits of each ending point, and then assuming a 8-bit or 9-bit index to the ending points, the 2 bytes assumption can be justified.

The optimal physical online time requires by the three tradeoff algorithms are summarized in Table 6.1, together with the corresponding parameter sets. In contrast with the expectation of the most programmers who practically implemented the perfect rainbow tradeoff so that the perfect rainbow tables are treated in serial, such as in [1] and [3], the practical online time to execute the parallel version of the perfect rainbow tradeoff is much shorter than that of the serial version with the same pre-computation investment. Also, the $\bar{\mathtt{pD}}$ tradeoff is even much faster than the $\bar{\mathtt{sR}}$ tradeoff for the online phase. More pre-computation effort is required for the $\bar{\mathtt{pD}}$ tradeoff, but complexity of the pre-computation phase is not a very important factor, as explained at the start of this chapter, also its difference is not very large to be concerned.

The online performance of the $\bar{\mathtt{sR}}$ tradeoff is not affected by the storage size to be used, once it is over 2.80 TB when $\mathsf{N} = 2^{52.574}$ and 99.9% success rate is required. This is because, as presented in Section 6.4, the physical online time taken by the $\bar{\mathtt{sR}}$ tradeoff is minimized at the certain pre-determined storage M, expressed by (6.5). However, both formulas of physical online times for the $\bar{\mathtt{pR}}$ and the $\bar{\mathtt{pD}}$ tradeoffs, expressed by (6.2) and (6.3), are in inverse proportion to the storage M. Hence, if the larger hard disk drive is available, the faster online phases can be achieved. Thus, one can claim that even though the physical time taken by access to the pre-computation table and search an identical ending point is fully taken into account to the tradeoff performance comparison, still the parallel versions of the perfect table tradeoffs, namely, the $\mathtt{p\bar{R}}$ and the $\mathtt{p\bar{D}}$ tradeoffs, give better performances on the online phases than the $\mathtt{s\bar{R}}$ tradeoff. In the case of the $\mathtt{p\bar{D}}$ tradeoff, it should be noted that the increase of the pre-computation, induced by use of larger storage amount and so setting the higher matrix stopping constant $\mathtt{p\bar{D}}_{\mathrm{msc}}$ to minimize the physical online time $T^{\mathtt{wc}}_{\mathtt{p\bar{D}}}$, may become a considerable level at some point.

Now, let us discuss when 4 TB SSD is available to perform the $\mathtt{s\bar{R}}$ tradeoff and each table entry of the perfect rainbow table is stored in $3+1=4$ bytes, in order to be advantageous to the $\mathtt{s\bar{R}}$ tradeoff. Since the implementations of the $\mathtt{s\bar{R}}$ tradeoff divide each pre-computation table into multiple sub-tables, by the considerations of using the bigger index files and the ending point truncation technique, the 4 bytes assumption can be possible. If 550 MB/s sequential read speed is assumed as be claimed by Samsung 850 PRO SATA-III 1 TB SSD as an optimal speed, then $\tau_L = \frac{4}{550\times 10^6} = 7.27\times 10^{-9}\,(\mathrm{sec})$ can be fixed. Under these assumptions, much advantageous to the $\mathtt{s\bar{R}}$ tradeoff, in the analogous arguments with the above, the minimal physical online time taken by the $\mathtt{s\bar{R}}$ tradeoff is $6.57\times 10^9 \times \tau_F$ with $M = 8.50\times 10^{11}$, so 3.40 TB storage requirement to achieve 99.9% success rate. Thus, even the $\mathtt{p\bar{R}}$ and the $\mathtt{p\bar{D}}$ tradeoffs use 4 TB HDD hard disk, the $\mathtt{s\bar{R}}$ tradeoff still falls behind the others.

We have seen an example of how the optimal physical online times for the three tradeoff algorithms are computed. More examples of these optimal performances can be seen in Table 6.2.

When $\mathsf{N} = 2^{52.574}$, at the 99% and 90% probabilities of success, the optimal online times for the $\mathtt{s\bar{R}}$ and $\mathtt{p\bar{R}}$ tradeoff algorithms can be achieved in the analogous arguments as the above. As for the $\mathtt{p\bar{D}}$ tradeoff at the 99% success rate, if assuming 5 bytes are allocated to record each table entry, then to achieve the optimal online time, $\log m = 22.3$ must be used with $\mathtt{p\bar{D}}_{\mathrm{msc}} = 5.56$, which requires $\vec{m}_0 = 2^{24.4}$ starting points. Thus, the 3 bytes assumption for each starting

Table 6.2: Optimal physical online times of $\bar{\mathtt{sR}}$, $\bar{\mathtt{pR}}$, and $\bar{\mathtt{pD}}$ tradeoffs at various situations.(4TB storage.)

N	ps	$\frac{\text{bytes}}{\text{sp+ep}}$	$\frac{\tau_L}{\tau_F}$	$\frac{\tau_H}{\tau_F}$	**online time**	pre-comp	alg.
$2^{52.574}$	99%	$5+2$	0.34		$2.99\times10^{10}\times\tau_F$	19.81N	$\bar{\mathtt{sR}}$
		$5+2$		5.10×10^4	$2.49\times10^{9}\times\tau_F$	19.81N	$\bar{\mathtt{pR}}$
		$3+2$		4.60×10^4	$3.15\times10^{9}\times\tau_F$	26.05N	$\bar{\mathtt{pD}}$
$2^{52.574}$	90%	$5+2$	0.34		$2.84\times10^{10}\times\tau_F$	5.43N	$\bar{\mathtt{sR}}$
		$5+2$		5.10×10^4	$1.35\times10^{9}\times\tau_F$	5.43N	$\bar{\mathtt{pR}}$
		$4+2$		4.77×10^4	$1.62\times10^{9}\times\tau_F$	14.05N	$\bar{\mathtt{pD}}$
		$3+2$		4.60×10^4	$1.97\times10^{9}\times\tau_F$	4.14N	$\bar{\mathtt{pD}}$
$2^{46.004}$	99.9%	$5+1$	0.32		$1.35\times10^{9}\times\tau_F$	50.59N	$\bar{\mathtt{sR}}$
		$5+2$		5.47×10^4	$2.71\times10^{7}\times\tau_F$	50.59N	$\bar{\mathtt{pR}}$
		$4+2$		5.12×10^4	$1.20\times10^{7}\times\tau_F$	1105.24N	$\bar{\mathtt{pD}}$
		$4+2$		5.12×10^4	$1.85\times10^{7}\times\tau_F$	50.59N	$\bar{\mathtt{pD}}$
		$3+2$		4.94×10^4	$3.55\times10^{8}\times\tau_F$	7.37N	$\bar{\mathtt{pD}}$
$2^{46.004}$	99%	$5+1$	0.32		$1.36\times10^{9}\times\tau_F$	19.81N	$\bar{\mathtt{sR}}$
		$5+2$		5.47×10^4	$2.02\times10^{7}\times\tau_F$	19.81N	$\bar{\mathtt{pR}}$
		$4+2$		5.12×10^4	$1.02\times10^{7}\times\tau_F$	290.85N	$\bar{\mathtt{pD}}$
		$4+2$		5.12×10^4	$1.65\times10^{7}\times\tau_F$	19.81N	$\bar{\mathtt{pD}}$
		$3+2$		4.94×10^4	$5.18\times10^{8}\times\tau_F$	4.74N	$\bar{\mathtt{pD}}$

point cannot be justified. Since $T^{\mathtt{wc}}_{\bar{\mathtt{pD}}}$ is a decreasing function until $\bar{\mathtt{pD}}_{\text{msc}} = 5.56$ at the fixed $M = \frac{4\times10^{12}}{5}$, the maximal $\bar{\mathtt{pD}}_{\text{msc}}$ value satisfying the number of starting point to be required $\vec{m}_0 < 2^{24}$ is $\bar{\mathtt{pD}}_{\text{msc}} = 4.13$, and the corresponding physical online time is $T^{\mathtt{wc}}_{\bar{\mathtt{pD}}} = 3.15\times10^9\times\tau_F$. In the similar way, at the 90% success rate, when each table entry is recorded in 5 bytes, the optimal online time is $T^{\mathtt{wc}}_{\bar{\mathtt{pD}}} = 1.97\times10^9\times\tau_F$ with $\bar{\mathtt{pD}}_{\text{msc}} = 0.94$. On the other hand, if assuming 6 bytes are allocated to record each table entry from the first, then $M = \frac{4\times10^{12}}{6}$ and the optimal physical online time is $T^{\mathtt{wc}}_{\bar{\mathtt{pD}}} = 1.62\times10^9\times\tau_F$, where

$$\tau_H^{\bar{\mathtt{pD}}} = 0.973\times10^{-2}.$$

is set from the measurement under 6 bytes assumption for perfect DP tables in Appendix A.

Now, examples for the key space size $\mathsf{N} = 2^{46.004}$ are following. At the first, the realistic figure τ_F is newly set as

$$\tau_F = 1.90 \times 10^{-7}\,\text{seconds},$$

by the measurement presented in Appendix A.

Let us first calculate the optimal physical online time for the $\bar{\mathtt{sR}}$ tradeoff. The 6 bytes assumption can be made from the fact $\mathsf{N}^{\frac{2}{3}} = 2^{30.7}$ and the storage optimization techniques. In this case, one can reset the realistic value

$$\tau_L = 7.01 \times 10^{-8} \times \frac{6}{7} = 6.01 \times 10^{-8}\,\text{seconds}.$$

At the 99.9% success rate, the optimal online time $T^{\mathtt{wc}}_{\bar{\mathtt{sR}}} = 1.35 \times 10^9 \times \tau_F$ can be obtained with $\log m = 32.2$, which justifies that each ending point can be recorded within only 1 bytes by the ending point truncation technique and use of index files. Likewise, the optimal online time can be computed for the $\bar{\mathtt{sR}}$ tradeoff at the 99% success rate.

As for the $\bar{\mathtt{pR}}$ tradeoff, by the similar work, 7 bytes allocation for each table entry is necessary for both 99.9% and 99% success rates, and the optimal online times can be easily computed and are presented in Table 6.2.

Now, let us finally compute the optimal online times for the $\bar{\mathtt{pD}}$ tradeoff. As before, under the 5 bytes assumption, the optimal online times are $T^{\mathtt{wc}}_{\bar{\mathtt{pD}}} = 3.55 \times 10^8 \times \tau_F$ and $T^{\mathtt{wc}}_{\bar{\mathtt{pD}}} = 5.18 \times 10^8 \times \tau_F$ at the 99.9% and 99% success rates, respectively. Now, let us assume that 6 bytes are allocated to record each table entry. Then, $T^{\mathtt{wc}}_{\bar{\mathtt{pD}}} = 1.20 \times 10^7 \times \tau_F$ and $T^{\mathtt{wc}}_{\bar{\mathtt{pD}}} = 1.02 \times 10^7 \times \tau_F$ can be achieved as the optimal online times at the 99.9% and 99% success rates. However, the pre-computation costs to achieve these optimal online times are $1105.24\mathsf{N} \times \tau_F$ and $290.85\mathsf{N} \times \tau_F$, which might be too large to be handled by tradeoff implementors, as compared with the fact that almost maximal pre-computation cost for the perfect rainbow tradeoff is $50.59\mathsf{N}$. Thus, at both 99.9% and 99% success rates, even though the optimal online times for the $\bar{\mathtt{pD}}$ tradeoff are shorter than those for the $\bar{\mathtt{pR}}$

tradeoff, one can say that the $\mathtt{p\bar{D}}$ tradeoff is hard to be preferred in practice against the $\mathtt{p\bar{R}}$ tradeoff. However, if observing the physical online time for the $\mathtt{p\bar{D}}$ tradeoff at the same pre-computation cost associated with the optimal online time achieved by the $\mathtt{p\bar{R}}$ tradeoff, the $\mathtt{p\bar{D}}$ tradeoff still provides the better performance than the $\mathtt{p\bar{R}}$ tradeoff. To be more precise, as seen in Table 6.2, $T^{\mathtt{wc}}_{\mathtt{p\bar{D}}} = 1.85 \times 10^7 \times \tau_F$ and $T^{\mathtt{wc}}_{\mathtt{p\bar{D}}} = 1.65 \times 10^7 \times \tau_F$ can be obtained by the pre-computatin efforts $50.59\mathsf{N}$ and $19.81\mathsf{N}$ under the 99.9% and 99% success requirements, respectively.

Therefore, when the search space size is $\mathsf{N} = 2^{46.004}$, the $\mathtt{p\bar{D}}$ tradeoff is likely to be preferred than the other two perfect rainbow tradeoff algorithms, to invert one-way function with high success rate. This can be meaningful conclusion, because the perfect rainbow tradeoff is well-known to be the best efficient tradeoff algorithm.

To conclude, in any case, the $\mathtt{s\bar{R}}$ tradeoff falls behind the parallel versions of the perfect rainbow and DP tradeoffs, and this is opposite to what the programmers of [1, 3] were expected.

Chapter 7

Conclusion

Our work can be divided into three parts. Firstly, we analyzed the execution complexity of the perfect DP, and perfect parallel DP tradeoffs and computed its time memory tradeoff coefficients. We also combined existing results on the execution complexity of the perfect rainbow tradeoff to present its online efficiency. In the analyses, we did not ignore a non-negligible cost for resolving false alarm of each tradeoff algorithm, and theoretical results were verified experimentally.

Secondly, using this information, the performances of the four DP variant algorithms, namely, the original DP, the parallel DP, the perfect DP, and the perfect parallel DP tradeoffs, were compared against each other. The previous results [18] and [16] on the original DP and the parallel DP tradeoffs were also included in our comparison, respectively. As a simplified conclusion, the perfect parallel DP tradeoff is advantageous over the other three DP tradeoffs in most cases. Hence, we compared the perfect parallel DP tradeoff with the non-perfect and the perfect rainbow tradeoffs. The result from the previous work [18] on the non-perfect rainbow tradeoff was also used in comparison. Our simplified conclusion on these comparisons was that the perfect rainbow tradeoff outperforms the perfect parallel DP and the non-perfect rainbow tradeoffs in typical situations. However, there may be special circumstances under which the preferences could be different. For example, importance of lowering the pre-computation cost may shift the preference towards the non-perfect rainbow tradeoff, and the need for flexibility on choosing parameter may make the perfect parallel tradeoff favorable at success rate of inversion less than 90%. In addition, we made the meaningful conclusion that the perfect parallel DP tradeoff could be more preferable over the non-perfect rainbow tradeoff unless one is extremely constrained in the amount of pre-computation possible. In these comparisons, we utilized the comparison method introduced by the previous work [18]. Both online execution complexity and cost for pre-computation of each tradeoff took into account, and known techniques to reduce storage size were also considered.

Thirdly, we computed the optimal physical online execution times of the perfect parallel DP $\mathtt{p\bar{D}}$, the perfect parallel rainbow $\mathtt{p\bar{R}}$, and the perfect serial rainbow $\mathtt{s\bar{R}}$ tradeoffs and compared against each other. Our work related to the optimal physical online time of the tradeoff algorithms was motivated by the recent work [21]. The paper noted that the online phase of the $\mathtt{s\bar{R}}$ tradeoff is roughly same as what is practically implemented by [1,3], even though the parallel processing of the rainbow tables was suggested by the algorithm designer [24]. The work [21] also analyzed the online execution complexity of the $\mathtt{s\bar{R}}$ tradeoff, and the analysis included not only time memory tradeoff coefficient, but also the expected number of table entries to be loaded into fast memory from hard disk drive. Using this information, we could express the physical online time of the $\mathtt{s\bar{R}}$ tradeoff and compute its minimum value, under a certain fixed success rate. In the similar manner, we could also express the total physical online execution times of the $\mathtt{p\bar{D}}$ and $\mathtt{p\bar{R}}$ tradeoffs, which include times taken for accessing to the pre-computed data, resided in slow hard disk. These extra costs for the online phases have been ignored in the previous theoretical approaches of the tradeoff algorithms.

We could conclude from the formulas that, for the $\mathtt{p\bar{D}}$ and $\mathtt{p\bar{R}}$ tradeoffs, at a fixed success rate, each storage size M determines a minimum physical online time and that this minimum time becomes smaller as the storage size M is increased. On the other hand, for the $\mathtt{s\bar{R}}$ tradeoff, there exists an M value that provides the minimum physical online time, so that, once the available storage size reaches this M value, the physical online time cannot be improved further. Each minimum physical online execution time could be calculated under the realistic constants set through our test measurements, after the realistic password space and high success rate of inversion were set. The $\mathtt{s}\bar{\mathtt{R}}$ tradeoff falls far behind the other two tradeoffs in comparison of minimum physical online time against each other, when 4TB HDD hard disk is assumed to be available. Also, one could claim that even though 4TB SSD was available, the conclusion, that is, two parallel tradeoffs' superiority to the $\mathtt{s}\bar{\mathtt{R}}$ tradeoff did not change.

When comparing between the $\mathtt{p}\bar{\mathtt{D}}$ and $\mathtt{p}\bar{\mathtt{R}}$ tradeoffs, the superiority depended on a search space size. For example, when the set of all passwords of lengths up to 8, constructed from all 95 characters on the standard keyboard is fixed to the search space of the tradeoff algorithms, the optimal online time for the $\mathtt{p}\bar{\mathtt{R}}$ tradeoff is smaller than that for the $\mathtt{p}\bar{\mathtt{D}}$ tradeoff at 99.9%, 99%, and 90% success requirements. However, when the aimed password lengths are replaced by up to 7, the physical online time of the $\mathtt{p}\bar{\mathtt{D}}$ tradeoff is smaller than the optimal physical online time of the $\mathtt{p}\bar{\mathtt{R}}$ tradeoff, even with the same pre-computation effort, under 99.9% and 99% success requirements.

It remains to extend this work and verify whether the lesser known recently proposed tradeoff algorithms [5, 14, 17, 23, 28–33] are superiority to the more widely used algorithms, in the sense considered in this work. Note that the results and approaches of this work have already been used to show [19, 20] that the fuzzy rainbow tradeoff [7, 8] could be seen as performing better than even the perfect rainbow tradeoff.

Bibliography

[1] (2015, Oct.) Free Rainbow Tables, Distributed Rainbow Table Project. [Online]. Available: http://freerainbowtables.com

[2] (2015, Oct.) Objectif Sécurité, Ophcrack. [Online]. Available: http://ophcrack.sourceforge.net

[3] (2015, Oct.)RainbowCrack Project. [Online]. Available : http://project-rainbowcrack.com

[4] (2015, Oct.) rcracki_mt. [Online]. Available: http://sourceforge.net/projects/rcracki/

[5] M. Ågren, T. Johansson, M. Hell, Improving the rainbow attack by reusing colours. In *CANS 2009*, LNCS **5888**, (Springer, 2009) pp. 362–378.

[6] G. Avoine, P. Junod, P. Oechslin, Characterization and improvement of time-memory trade-off based on perfect tables. *ACM Trans. Inform. Syst. Secur.* **11**(4), 17:1–17:22 (2008). Preliminary version presented at INDOCRYPT 2005.

[7] E. P. Barkan, Cryptanalysis of Ciphers and Protocols. Ph.D. Thesis, Technion—Israel Institute of Technology, March 2006.

[8] E. Barkan, E. Biham, A. Shamir, Rigorous bounds on cryptanalytic time/memory tradeoffs. In *Advances in Cryptology—CRYPTO 2006*, LNCS **4117**, (Springer, 2006), pp. 1–21.

[9] A. Biryukov, A. Shamir, D. Wagner, Real time cryptanalysis of A5/1 on a PC. In *FSE 2000*, LNCS **1978**, (Springer, 2001), pp. 1–18.

[10] J. Borst, *Block Ciphers: Design, Analysis, and Side-Channel Analysis*. Ph.D. Thesis, Katholieke Universiteit Leuven, September 2001.

[11] J. Borst, B. Preneel, J. Vandewalle, On the time-memory tradeoff between exhaustive key search and table precomputation. In *Proceedings of the 19th Symposium on Information Theory in the Benelux*, (WIC, 1998), pp. 111–118.

[12] A. Fiat, M. Naor, Rigorous time/space tradeoffs for inverting functions. In *Proceedings of the twenty-third annual ACM Symposium on Theory of Computing*, (ACM, 1991), pp. 534–541.

[13] M. E. Hellman, A cryptanalytic time-memory trade-off. *IEEE Trans. on Infor. Theory* **26**, pp. 401–406 (1980).

[14] Y. Z. Hoch, Security analysis of generic iterated hash functions. Ph.D. Thesis, Weizmann Institute of Science, August 2009.

[15] J. Hong, The cost of false alarms in Hellman and rainbow tradeoffs. *Des. Codes Cryptogr.* **57**(3), pp. 293–327 (2010).

[16] J. Hong, G. W. Lee, D. Ma, Analysis of the parallel distinguished point tradeoff. In *Progress in Cryptology—INDOCRYPT 2011*, LNCS **7107**, (Springer, 2011), pp. 161–180.

[17] J. Hong, K. C. Jeong, E. Y. Kwon, I.-S. Lee, D. Ma, Variants of the distinguished point method for cryptanalytic time memory trade-offs. In *ISPEC 2008*, LNCS **4991**, (Springer, 2008), pp. 131–145.

[18] J. Hong, S. Moon, A comparison of cryptanalytic tradeoff algorithms. *J. Cryptology* **26**(4), pp. 559—637 (2013).

[19] B.-I. Kim, J. Hong, Analysis of the non-perfect table fuzzy rainbow tradeoff. In *ACISP 2013*, LNCS **7959**, (Springer, 2013), pp. 347–362.

[20] B.-I. Kim, J. Hong, Analysis of the perfect table fuzzy rainbow tradeoff. *J. Appl. Math.*, **2014**, (2014), Article ID 765394.

[21] J. W. Kim, J. Hong, K. Park, Analysis of the rainbow tradeoff algorithm used in practice. Cryptology ePrint Archive, Report 2013/591.

[22] G. W. Lee, J. Hong, Comparison of perfect table cryptanalytic tradeoff algorithms. To appear in *Des. Codes Cryptogr.* (Online July 2015), http://dx.doi.org/10.1007/s10623-015-0116-0

[23] S. Mukhopadhyay, P. Sarkar, Nearly orthogonal rainbow tables. Indian Statistical Institute Technical Report, No. ASD/2004/9, November 2004.

[24] P. Oechslin, Making a faster cryptanalytic time-memory trade-off. In *Advances in Cryptology—CRYPTO 2003*, LNCS **2729**, (Springer, 2003), pp. 617–630.

[25] J.-J. Quisquater, J. Stern, Time-Memory Tradeoff Revisited. Unpublished, December 1998.

[26] A. Shamir, Random Graphs in Security and Privacy. Invited talk at ICITS 2009.

[27] F.-X. Standaert, G. Rouvroy, J.-J. Quisquater, J.-D. Legat, A time-memory tradeoff using distinguished points: New analysis & FPGA results. In *Cryptographic Hardware and Embedded Systems—CHES 2002*, LNCS **2523**, (Springer, 2003), pp. 593–609.

[28] V. Thing, Virtual expansion of rainbow tables. In *Advances in Digital Forensics VI*, IFIP AICT 337, pp.243–256, (2010).

[29] V. L. L. Thing, H.-M. Ying, A novel time-memory trade-off method for password recovery. *Digital Investigation* **6**, pp. S114–S120 (2009).

[30] V. L. L. Thing, H.-M. Ying, Rainbow table optimization for password recovery. *International Journal on Advances in Software* **4**, pp. 479–488, (2011).

[31] W. Wang, D. Lin, Z. Li, T. Wang, Improvement and analysis of VDP method in time/memory tradeoff applications. In *ICICS 2011*, LNCS **7043**, (Springer, 2011), pp. 282–296.

[32] H.-M. Ying, V. L. L. Thing, A novel rainbow table sorting method. In *CYBERLAWS 2011*, pp.35–40, (2011).

[33] W. Zhang, M. Zhang, Y. Liu, R. Wang, A new time-memory-resource tradeoff method for password recovery. In *ICCIIS 2010*, (IEEE, 2010), pp. 75–79.

Appendix A

Practical System Constants τ_F, τ_L, and τ_H

In this chapter, we will describe our measurements to decide realistic constants τ_F, τ_L, and τ_H. Our machine consisted of an Intel Core i7-3770K 3.50GHz octa-core CPU, a 16GB DDR3 main memory, and a 4TB 7200RPM SATA hard disk drive.

A.1 τ_F

Recalling that τ_F is the physical time taken by a single iteration of the *colored* one-way function, the physical time taken by the reduction process, which converts hash value into index, should be taken into account to the measurement. In details, the colored one-way function, which we treated, is as follows, together with the cryptographic hash function MD5 as the one-way function to be inverted.

$$\begin{tikzcd}
\underset{\text{index}}{\bullet} \arrow[r] & \underset{\text{password}}{\circ} \arrow[r, "\text{hash function}", "\text{MD5}"'] & \underset{\text{hash}}{\circ} \arrow[r, "\text{reduction}"] & \underset{\text{index}}{\bullet}
\end{tikzcd}$$

To set the realistic figure τ_F, we downloaded the online phase program `rcracki_mt` [4], together with two pre-computation tables from [1]. Each of those two is the di-

Table A.1: Measurement results for the system constant τ_F

N	# of chain walks / a hash	# of hashes	total time(sec)	τ_F(s/iter.)
$2^{52.574}$	19999700001	5	20395.17	2.04×10^{-7}
$2^{46.004}$	799940001	10	1520.21	1.90×10^{-7}

vided sub-table of a single perfect rainbow table working with the search space $\mathsf{N} = 2^{52.574}$ or $\mathsf{N} = 2^{46.004}$. The search space sizes $\mathsf{N} = 2^{52.574}$ and $\mathsf{N} = 2^{46.004}$ correspond to the set of all passwords of lengths up to 8 and 7, constructed from all 95 characters on the standard keyboard, respectively. Then, using the program `rcracki_mt`, we measured the physical time taken by the online chain creations for 5 or 10 given MD5 hashes. The measured results of our machine are in Table A.1.

Recall that for the $\mathtt{s}\bar{\mathtt{R}}$ tradeoff, all t online chains, which correspond to a single perfect rainbow table, are generated at once before search merges between those online ending points and the pre-computed ending points in the table, where t is the length of a rainbow chain. When the search space size $\mathsf{N} = 2^{52.574}$, for a single MD5 hash, 19999700001 invocations of the colored one-way function are required to create all online chains which corresponds to a single perfect rainbow table, and our machine took 20395.17 seconds to perform the online chain creations for 5 MD5 hashes. To be more precise, the downloaded pre-computation table from [1] consists of rainbow chains of length $t = 199999$. Since each online chain is assumed to start with the unknown answer, input to the given password hash, the number of required colored one-way function invocations to generated all t online chains is given by

$$\sum_{i=0}^{t-1} i = 19999700001.$$

This amount of required invocations can be also found when the online program `rcracki_mt` is run under its debug mode. Likewise, when the search space size is $2^{46.004}$, the chain length t is set as 39999 and it requires 799940001 one-way function invocations to create all online chains. Remark that [1] treated the chain length as the number of points included in a single rainbow chain, so that 200000 and 40000 were recorded in those corresponding tables as chain lengths.

Averaging the measuring results, we found $\tau_F = 2.04 \times 10^{-7}$ (sec/iteration) and $\tau_F = 1.90 \times 10^{-7}$ (sec/iteration) for the search space size $\mathsf{N} = 2^{52.574}$ and $\mathsf{N} = 2^{46.004}$, respectively.

A.2 τ_L

As was the previous section, we used the online phase program `rcracki_mt` [4] and downloaded the pre-computation tables from [1], correspond to the divided sub-table of the perfect rainbow table working with the search space size $\mathsf{N} = 2^{52.574}$. Each sub-table consists of 2^{26} table entries, and 7 bytes are allocated to record each table entry. We measured the total time taken by loading the 10 sub-tables from our hard disk drive to main memory, and it took 4702.60 seconds. Thus, averaging over 10 sub-tables and 2^{26} table entries, we found that

$$\tau_L = 7.01 \times 10^{-8}\,\text{seconds/entry},$$

when each table entry is recorded in 7 bytes.

We note that our measurement do not cover the time taken by reading the header of each table, which is irrelevant to loading of table entries, and only cover the time taken by reading the table entries and combining those with the index file information, which is brought already, to expand each table entry into 16 bytes complete form.

A.3 τ_H

Our measurements to set the realistic figure τ_H were made under the assumption that 4 TB hard disk drive is fully used to perform the online phase of $\bar{\mathtt{pR}}$ or $\bar{\mathtt{pD}}$ tradeoff algorithm.

Let us measure the time taken by a single search of the pre-computation perfect rainbow table stored in the hard disk drive for a matching ending point, when each table entry is recorded in 7 bytes. We created a random table which consisted of $2^{36.8}$ table entries and each starting point and ending point were recorded in 5 and 2 bytes, respectively. More precisely, a random 39 bits number was generated and regarded as the ending point after truncation, together with a sequentially increasing number, regarded as a starting point to generate the perfect table. Our random table consisted of $2^{36.8}$ table entries after sorting on the ending points and discarding merging entries. By using sequential starting points and applying 26-bit indices to 39-bit ending points, 7 bytes record of table entry is feasible.

Now, given a randomly generated 39-bit target, we calculated how many ending points having the same 26-bit index with the target are in the table, using the index file information. Then using that value and 13-bit tail of the target, together with index file information, we predicted a target's position in the table, and loaded 100 adjacent table entries into fast main memory which are the 50 table entries each, the front and the rear of the position. We checked that a merging ending point with the target is in the loaded 100 table entries using the binary search and returned the corresponding starting point if a match was found. If a match did not be found among loaded 100 table entries and a possibility to be found still existed, then all the possible remaining table entries were loaded into fast main memory and searched a match.

Results of measuring the execution speeds of the above processes, averaged over 20 trials for 10000 randomly generated targets each are in Table A.2. Measurement results can be separated into two parts, when a match was found and did

Table A.2: Measurement results for the system constant τ_H when a table entry is recorded in 7 bytes

$\tau_H^{\bar{\mathtt{pR}}}$	$\tau_H^{\bar{\mathtt{pR}}}$(success)	$\tau_H^{\bar{\mathtt{pR}}}$(faill)
0.010397 seconds	0.010390 seconds	0.010399 seconds

not be found. As be seen in Table A.2, since difference between success and fail speeds is negligible, we can choose realistic figure

$$\tau_H^{\bar{\mathtt{pR}}} = 1.04 \times 10^{-2}\,\text{seconds}$$

as the physical time taken by a single search of the perfect rainbow table stored in hard disk drive for a matching ending point when 7 bytes are allocated to record each table entry.

Let us now focus on the physical time taken by a single search of the perfect DP tables stored in the hard disk drive when each table entry is recorded in 5 or 6 bytes. Under the 5 bytes record situation, we assumed that about 310000 perfect DP tables are separately stored into 8 files and each table consists of $2^{21.3}$ table entries, recorded in 5 bytes each. Then each file contains $310000/8 = 38750$ perfect DP tables and requires approximately 500 GB. Since slightly more than 21.3 bits are required to record each ending point by the distinguished part removal and the ending point truncation techniques, 5 bytes assumption is justified, after applying the index file. For our construction, we randomly generated a 22-bit point as a truncated ending point and recorded a 14-bit ending point tail in 2 bytes assuming a 8-bit index to the sorted ending points. By use of sequential starting points, each starting point could be recorded in 3 bytes.

We made the two files, as described above and for randomly generated table index and ending point target, we searched a matching ending point in the table corresponding to the given table index, and returned the corresponding starting point or fail, in the analogous way to the measurement for a perfect rainbow table. A single difference with the previous measurement is that 50 adjacent table entries to a predicted position are loaded, not 100 adjacent entries. The speed of the table lookup and searching process, averaged over 20 trials for 10000 randomly generated targets each is

$$\tau_H^{\bar{\mathtt{pD}}} = 0.939 \times 10^{-2}\,\text{seconds.}$$

For the 6 bytes record situation, we assumed that about 43600 perfect DP tables are separately stored into 8 files and each table consists of $2^{23.8}$ table entries, recorded in 6 bytes. We made two files so that each file consisted of 5450 random tables, and a starting point and an an ending point were recorded in 4 and 2 bytes, respectively. More precisely, use of a 11-bit index to a 25-bit truncated ending point allowed us to record only 14-bits ending point tail in 2 bytes. Measurement is very analogous to that of 5 bytes situation. The speed of the table lookup and searching process, averaged over 20 trials for 10000 randomly generated targets each is

$$\tau_H^{\bar{\mathtt{pD}}} = 0.973 \times 10^{-2}\,\text{seconds.}$$

국문초록

최근 발표된 논문에서 Hellman의 시간 저장공간 절충기법과 중복가능 테이블을 이용한 특이점 절충기법, 레인보우 테이블 기법이 분석되었고, 서로간의 성능비교 또한 수행된 바 있다. 그 분석은 오경보를 해결하는 추가비용까지 고려해 분석이 정확하도록 하였고, 알고리즘 성능비교에는 온라인 수행복잡도와 사전계산 비용을 함께 고려할 뿐만아니라 저장공간의 최적화 기술 또한 적용하여 공평하도록 하였다. 이 논문을 토대로 또다른 최근 논문에서 중복가능 특이점 테이블을 병렬처리하는 절충기법의 분석 및 알고리즘간 비교 또한 수행된 바 있다. 본 연구에서는 그밖의 세가지의 절충기법의 성능을 분석하고, 앞서 언급한 네가지 절충기법과 수행능력을 비교한다. 새로이 분석할 알고리즘은 중복제거 테이블을 직렬 또는 병렬처리하는 두가지 특이점 절충기법과 중복제거 테이블을 병렬처리하는 레인보우 테이블 기법을 포함한다.

알고리즘의 성능은 하나의 숫자로 표현될 수 없고, 그 선호도 또한 알고리즘 사용자의 구현환경에 의존한다. 그렇기 때문에, 우리는 절충기법의 성능을 알고리즘의 사용가능한 파라미터들에 의한 곡선으로 표현하여 구현자가 선택할 수 있도록 하였다. 그러나 우리는 보편적인 상황에서 중복제거 특이점 테이블을 병렬처리하는 절충기법이 다른 특이점 절충기법들 중 가장 우수한 성능을 보이고, 중복제거 레인보우 테이블을 이용한 절충기법이 모든 우리가 고려한 절충기법들 중 가장 뛰어나다는 결론을 내렸다.

다른 한편으로, 최근 논문에서 레인보우 테이블기법을 최초 고안한 논문에서 테이블을 병렬처리하는 방법을 제시한 것에 반하게 실제로는 중복제거 레인보우 테이블을 직렬적인 방식으로 처리한다는 사실을 언급한 바 있다. 그 이유는 사전계산 테이블의 병렬처리가 이론적으로는 더 효율적이나 패스워드 크랙킹과 같이 실제 상황에 적용되기 위해서는 테이블의 크기가 매우 커 빠른 주메모리에 테이블을 모두 올려 병렬처리하는 것이 불가능하기 때문이다. 이와 같은 접근방식으로 우리는 중복제거 테이블을 직렬처리

또는 병렬처리하는 레인보우 절충기법과 중복제거 특이점 테이블을 병렬처리하는 절충기법의 최적의 물리적 수행시간을 계산했다. 이 계산은 다양한 실제적인 패스워드 공간들과 다양한 높은 수준의 성공확률이 요구되는 상황에서 디스크 용량에 특정 제한이 있는 경우를 가정한 후 수행하였다. 물리적인 온라인 단계 수행시간의 계산에는 기존의 이론적인 접근방식들에서는 무시했던, 빠른 주메모리로 테이블을 로딩하는 시간 또는 느린 디스크로 테이블 검색을 수행하는 시간을 고려하였다.

결과적으로 우리는 실제적 알고리즘 구현자의 감각과는 다르게, 알고리즘들의 최적화된 물리적 수행시간을 비교하였을 때, 직렬처리를 적용한 중복제거 레인보우 절충기법이 다른 두 가지 병렬처리 알고리즘에 비해 많이 뒤떨어진다는 것을 알아냈다. 단순하게 내릴 수 있는 결론으로, 우리가 다룬 두가지 패스워드 공간 중 더 큰 공간에서는 중복제거 레인보우 테이블을 병렬처리한 절충기법이 가장 빠르게 온라인 단계를 수행하고 더 작은 공간에서는 중복제거 특이점 테이블을 병렬처리한 절충기법이 가장 빠르게 온라인 단계를 수행한다는 것을 알 수 있었다.

주요어휘: 시간 저장공간 절충기법, 특이점, 레인보우 테이블, 중복제거 테이블, 알고리즘 수행복잡도

학번: 2011-30898