Des. Codes Cryptogr. (2016) 80:473–523
DOI 10.1007/s10623-015-0116-0

Comparison of perfect table cryptanalytic tradeoff algorithms

Ga Won Lee[1] · **Jin Hong**[1]

Received: 25 September 2014 / Revised: 15 June 2015 / Accepted: 7 July 2015 /
Published online: 26 July 2015

Abstract The performances of three major time memory tradeoff algorithms were compared in a recent paper. The algorithms considered there were the classical Hellman tradeoff and the non-perfect table versions of the distinguished point method and the rainbow table method. This paper adds the perfect table versions of the distinguished point method and the rainbow table method to the list, so that all the major tradeoff algorithms may now be compared against each other. Even though there are existing claims as to the superiority of one tradeoff algorithm over another algorithm, the algorithm performance comparisons provided by the current work and the recent paper mentioned above are of higher practical value. We provide comparisons of algorithms at parameters that achieve a common success rate of inversion and which take both the cost of pre-computation and the efficiency of the online phase into account. The comparisons are based on the average case execution behaviors rather than the worst case situations, and non-negligible details such as the effects of false alarms and various storage optimization techniques are no longer ignored. A large portion of this paper is allocated to analyzing the execution behavior of the perfect table distinguished point method. In particular, we obtain a closed-form formula for the average length of chains associated with a perfect distinguished point table.

Keywords Time memory tradeoff · Distinguished point · Rainbow table · Perfect table · Algorithm complexity

Mathematics Subject Classification 94A60 · 68W40

Communicated by C. Cid.

✉ Jin Hong
jinhong@snu.ac.kr

Ga Won Lee
gwlee87@snu.ac.kr

[1] Department of Mathematical Sciences, Seoul National University, Seoul 151-747, Korea

1 Introduction

A cryptanalytic time memory tradeoff algorithm is a method for inverting one-way functions with the help of pre-computed data. It is widely used by hackers and law enforcement authorities to recover passwords from the knowledge of password hashes. In the pre-computation phase, a massive amount of computation specific to the one-way function of interest is performed and a compact digest of the obtained information is stored as tables. When the target image for inversion is given, further computations that utilize the pre-computed tables are performed to recover the input used to create the target with a reasonable probability, and this part is referred to as the online phase.

The execution behavior of any tradeoff algorithm can be controlled through its many parameters. Existing analyses [2,4,10,11] show that many tradeoff algorithms satisfy the tradeoff curve

$$TM^2 = c\,\mathsf{N}^2, \tag{1}$$

for some numeric value c, which is neither very large nor close to zero. Here, T is the online execution time, M is the size of the memory space required to store the pre-computation tables, and N is the size of the space the one-way function is acting on. In particular, if the online phase of a tradeoff method executes in time T using tables of combined size M under some set of its parameters, then given any other T' and M' such that $TM^2 = T'M'^2$, there exists another set of parameters under which the algorithm will execute in time T' using storage M', thus allowing for tradeoffs to be made between the online execution time and the storage requirement.

There are many time memory tradeoff methods available today, with most of them having roots in the classical method by Hellman [11]. The most widely known methods are the distinguished point (DP) variant of the Hellman's original algorithm [7,8] and the rainbow table method [21], which we shall refer to in this paper as the *DP tradeoff* and the *rainbow tradeoff*, respectively. Both of these algorithms have two subversions that work with the non-perfect tables and the perfect tables.

Comparison of tradeoff algorithm performances has been a controversial subject, with every newly announced algorithm claiming superiority over existing algorithms. The difficulty in accurately analyzing the execution behavior of these algorithms is clearly one reason for this confusion, but another source has been the absence of an acceptable method for numerically presenting the performances of tradeoff algorithms in a manner that closely reflects our intuition concerning their relative usefulness or practicality.

Let us take a moment to explain a reasonable method of tradeoff algorithm comparison that was recently suggested by [14]. Notice that the tradeoff curve (1) corresponding to any specific tradeoff algorithm presents the complete list of (T, M)-pair options that are made available by the algorithm. Thus the tradeoff curve expresses the required online resources or the online execution behavior of an algorithm completely, and one may accept the *tradeoff coefficient* $c = \frac{TM^2}{\mathsf{N}^2}$ as a good measure of how efficient an algorithm is, with a smaller coefficient indicating a more efficient algorithm. Indeed, many previous claims as to the superiority of one algorithm over another have focused on this value.

However, the tradeoff coefficient alone cannot fully capture our intuition as to how good an algorithm is. Since it is obvious that the online time can always be reduced if one is willing to accept a lower success rate of inversion, the tradeoff coefficient is sure to change with the requirement on the inversion success rate. Furthermore, even when one is aiming for a fixed success rate, it is only reasonable to anticipate a more efficient online phase, or, equivalently, a smaller tradeoff coefficient, after a larger investment in the pre-computation phase.

In short, the online efficiency of each algorithm can be expressed succinctly through the tradeoff coefficient, but each algorithm allows further tradeoffs to be performed between the online efficiency, pre-computation effort, and success rate, while what we intuitively feel as the practicality or usefulness of an algorithm is directly connected to the overall behavior of the algorithm concerning these upper level tradeoffs. The difficulty of algorithm comparison lies in that, unlike the tradeoffs between time and storage that may commonly be expressed in the form (1) for most tradeoff algorithms, the equations that express the tradeoffs between the three aforementioned factors are very different among the major tradeoff algorithms. Hence, no single numeric value that can be computed for all algorithms in a common manner is likely to capture the performances of the algorithms concerning the upper level tradeoffs.

The solution suggested by [14] was to let the algorithm implementers make the final judgement and choice based on their requirements, available pre-computation and online resources, and personal taste, and to only present the information necessary for this decision in a coherent manner. Parameters for different algorithms are first restricted to those that achieve the same success rate. Then the tradeoffs between pre-computation cost and tradeoff coefficient are presented as curves for each algorithm. Each curve represents the complete array of options provided by one algorithm as to what degree of online efficiency can be obtained after a certain amount of pre-computation investment, at the specified success rate. Implementers that place different relative values on the online efficiency and the pre-computation cost will choose to use different algorithms. In fact, comparisons of the algorithms themselves are no longer meaningful, and each implementer will choose an algorithm together with the online efficiency and pre-computation cost pair made available by that algorithm, based on his or her favored balance between the two factors, from among all the options made available by all the algorithms.

The work [14] first computed the success rates, pre-computation costs, and accurate tradeoff coefficients for the classical Hellman, non-perfect DP, and non-perfect rainbow tradeoffs. These complexities and properties were presented as functions of the algorithm parameters. Then, after fixing a small number of specific success rates of interest, parameters were restricted to those achieving these success rates, and the upper level tradeoffs between the pre-computation cost and tradeoff coefficient were presented as curves, separately for each algorithm. After carefully adjusting the units expressing the tradeoff coefficients for the three algorithms into one directly comparable unit, the three curves were gathered together in one figure. This comprehensive and coherent display of information can allow someone considering the use of the tradeoff algorithms to decide on the most desirable balance between pre-computation investment and online efficiency from among the numerous options made available by the three tradeoff algorithms, at any required success rate.

The current paper completes the task started in [14] by dealing with the two remaining major tradeoff algorithms, namely, the perfect DP and perfect rainbow tradeoffs. We compute the tradeoff coefficients for these two algorithms and present the upper level tradeoffs between pre-computation cost and online efficiency as graphs at a small number of fixed success rates. Similar graphs for any other success rate may easily be obtained from our formulas. An overly simplified conclusion that may be drawn from the graphs is that, under most practical situations, the perfect rainbow tradeoff is likely to be preferred over the other four algorithms that have been mentioned. However, as we have discussed, the final judgment is not ours to make, and may be different under each specific situation.

Since this work is a direct extension of the work [14], we shall not repeat the contents of [14] that advocate the subject of our study. Although not strictly necessary, a good understanding of [14] will be very helpful in reading this paper. In fact, it should be possible for the impatient

reader with a full understanding of [14] to jump straight to Fig. 4 and collect the core findings of this paper.

We clarify that we will not be dealing with the time memory *data* tradeoffs [5,6] in this paper. Also note that even though one could conceive of a perfect table version of the classical Hellman tradeoff [2,29], it is never used in practice due to its impractical pre-computation phase and will not be treated in this work.

The rest of this paper is organized as follows. In the next section, we fix the terminology, clarify the exact versions of the algorithms we are analyzing, and review existing analyses of the perfect DP and perfect rainbow tradeoffs. Section 3, which is the most technical part of this paper, is devoted to analyzing the execution behavior of the perfect DP tradeoff. Analysis of the expected online time complexity that does not ignore false alarms is given and the tradeoff coefficient is computed. The issue of storage optimization is also discussed and tests that give strength to the correctness of our theoretical developments are presented. Since some readers may be under the impression that existing works already provide a full analysis of the perfect DP tradeoff, we also discuss why these previous results were insufficient for the purpose of algorithm comparisons. The perfect rainbow tradeoff is treated in Sect. 4. There were previous results we could utilize and obtaining the tradeoff coefficient for the perfect rainbow tradeoff was much easier than for the perfect DP tradeoff. The information we have prepared concerning the perfect DP and perfect rainbow tradeoffs is presented in Sect. 5 as graphs that allow direct comparisons between different algorithms and also between different parameter sets for the same algorithm. Finally, the paper is summarized in Sect. 6. The appendices contain further material that could be of interest.

2 Preliminaries

In this section, after recalling the various notions required for our discussion, we review some of the existing related works. Only the theoretical developments concerning the accurate time and storage analyses of the perfect DP and the perfect rainbow tradeoffs are explained. Some of the contents we do not present would include other tradeoff algorithms and implementation issues. There are also theories concerning asymptotic complexity bounds [4] on a general class of tradeoff algorithms and analyses of the full costs [32] of many cryptographic attack algorithms. We acknowledge that even the papers we introduce contain much more content than what is explained here.

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 [14] for more detail. The work [14] also clarifies many obscure technical details[1] 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

[1] Let us mention just one example. The objective of any tradeoff algorithm discussed in this work will be to recover the randomly chosen specific input that was used to create the given inversion target, rather than to recover any pre-image corresponding to the inversion target. This detail, which the previous sentence has *not* explained in full, is important when attempting an analysis of the accuracy aimed for in this work. However, even this most basic definition of a successful inversion is seldom made clear in the tradeoff literature.

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 section 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 Perfect DP tradeoff

As there are many small tweaks that can be applied to the DP tradeoff, we must clarify the exact version of the DP tradeoff that will be analyzed in this work. In short, we will be treating the perfect table variant that uses a sufficiently large chain length bound, utilizes the online chain recording technique to reduce the effort of resolving alarms, and employs the ending point truncation technique for storage reduction.

To setup the perfect table version of the DP tradeoff, one first fixes positive integer parameters m, t, and ℓ. In time, we will see that these correspond to the number of entries per table, the average length of a random DP chain, and the number of tables, respectively. 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 $\mathscr{N}$ with probability $\frac{1}{t}$. This *distinguishing property* must be extremely easy to check for elements of $\mathscr{N}$. Any element of $\mathscr{N}$ 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 $\mathscr{N}$ with $\log t$ leading zeros as DPs. Next, bijections $rd_i : \mathscr{N} \to \mathscr{N}$ $(i = 1, \ldots, \ell)$ 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, is fixed.

The pre-computation and online phases of the perfect DP tradeoff, in their rudimentary forms, are given by Algorithms 1 and 2, 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.

The integer $\vec{m}_0$ of Algorithm 1 must be set so that each table $\bar{\mathtt{DT}}_i$ is expected to contain m entries. The appropriate value for $\vec{m}_0$ is given later in this paper by Lemma 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 1 is called the i-th *perfect DP table* and the larger auxiliary table $\mathtt{DT}_i = \{(\mathtt{sp}_j^i, \mathtt{ep}_j^i)\}_{j=1}^{\vec{m}_0}$ 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

Algorithm 1: Pre-computation phase of the perfect DP tradeoff

for $i = 1, \ldots, \ell$ **do**
  $\mathtt{TDT}_i \leftarrow \emptyset$; // temporary DP table
  for $j = 1, \ldots, \vec{m}_0$ **do**
    Choose $\mathtt{sp}^i_j \in \mathscr{N}$; // 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* (sp, ep, 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 2: Online phase of the perfect DP tradeoff

for $i = 1, \ldots, \ell$ **do**
  $\mathtt{op} \leftarrow rd_i(\mathbf{h}) = F_i(\mathbf{p})$;
  while op *is not a DP* **do** // generate online chain
    $\mathtt{op} \leftarrow F_i(\mathtt{op})$;
  end
  Search for op among the 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 $[\mathtt{tp}_{\mathrm{new}} != rd_i(\mathbf{h})]$ *and* $[\mathtt{tp}_{\mathrm{new}}$ *is not a DP*$]$ **do** // regenerate pre-comp chain
      $\mathtt{tp}_{\mathrm{old}} \leftarrow \mathtt{tp}_{\mathrm{new}}$;
      $\mathtt{tp}_{\mathrm{new}} \leftarrow F_i(\mathtt{tp}_{\mathrm{old}})$;
    end
    if $F(\mathtt{tp}_{\mathrm{old}}) == \mathbf{h}$ **then** // answer found
      return $\mathtt{tp}_{\mathrm{old}}$ as answer and terminate;
    end
  end
end

will almost always be to the tables $\bar{\mathtt{DT}}_i$ and $\mathtt{DT}_i$ that result from an execution of Algorithm 1. We will not be discussing properties of perfect DP tables created in any other manner.

A series of elements from $\mathscr{N}$, 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 **p**, rather than from the inversion target **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 paper are not interchangeable, and the reader should exercise care to distinguishing the two. The matrices corresponding to

$\bar{\mathtt{DT}}_i$ and $\mathtt{DT}_i$ will be referred to as the *perfect DP matrix* $\bar{\mathtt{DM}}_i$ and the *non-perfect DP matrix* $\mathtt{DM}_i$, respectively. Once again, our use of these terms will almost always be to DP matrices resulting from an execution of Algorithm 1, rather than to matrices of perfect or non-perfect properties obtained in some other manner.

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.

As the online chain corresponding to a pre-computation table is generated, the chain will either reach a DP that does not reside in $\bar{\mathtt{DT}}_i$ or merge into one of the pre-computation chains of $\bar{\mathtt{DM}}_i$. Because the function F_i that is being iterated is not injective, the discovery of $\mathtt{op} == \mathtt{ep}_j^i$ 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 $\mathtt{sp}_j^i$, and these instances are called *false alarms*. Any ending point match $\mathtt{op} == \mathtt{ep}_j^i$ is an *alarm*, and the bottom half of Algorithm 2 works to *resolve* this alarm.

The collision removal process of Algorithm 1 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 [7,8] 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.

The starting points $\{\mathtt{sp}_j^i\}_{j=1}^{\vec{m}_0}$ for a pre-computation matrix must be chosen to be distinct. We specify more concretely that, within each table, sequential starting points [2,6,7] 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 $\mathscr{N}$.

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 [7,8]. That is, the while-loop of Algorithm 1 and the first while-loop of Algorithm 2 need to be augmented with another condition to prevent occurrences of infinite loops. A lower bound $\check{t}$ can also be used [22,25] to discard short pre-computation chains that contribute little to the search space coverage. In this work, 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. A brief justification as to why treating just this case is sufficient is given in Appendix 1, and a detailed explanation of what is meant by a sufficiently large upper bound on the chain length is given in Appendix 2.

The work [7,8] suggested that the length of each pre-computation chain and the maximum pre-computation chain length for each table be recorded in the DP table. However, the recording of individual chain lengths has a negative effect on the physical amount of required storage, and we can argue heuristically that the positive effect of the maximal chain length information is very limited. The DP tradeoff considered in this work incorporates neither of the two suggestions.

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 [6]. In addition, the ending points are

further truncated to a certain length before being written to storage [4,6]. 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. Details are discussed later in this paper.

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 [6], 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.

We assume that Algorithm 2 is slightly modified so as to incorporate the online chain record [12,24] technique. 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.

The work [12,24] suggested that all the pre-computation tables be processed in parallel, rather than sequentially, during the online phase. For the case of the non-perfect DP tradeoff, this idea was shown to have a small positive effect [16]. However, the parallel version of the perfect DP tradeoff will not be analyzed in this work. Treatment of parallelization is quite outside the scope of this work, but our work will be an important stepping stone for anyone interested in analyzing the parallel version. Some comments on this issue are given in Appendix 1.

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 $\bar{\mathtt{DT}}$, $\mathtt{DT}$, $\bar{\mathtt{DM}}$, 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.

The *coverage rate* $\bar{\mathtt{D}}_{\text{cr}}$ of a perfect DP matrix $\bar{\mathtt{DM}}$ is defined to be the expected number of distinct nodes $|\bar{\mathtt{DM}}|$ in the matrix, divided by mt. 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 $|\bar{\mathtt{DM}}|$ and $\bar{\mathtt{D}}_{\text{cr}}\frac{mt}{\mathsf{N}}$ is the success probability associated with a single perfect DP table.

We will use the notation $\bar{\mathtt{D}}_{\text{msc}} = \frac{mt^2}{\mathsf{N}}$ for the perfect DP tradeoff and refer to this as the *matrix stopping constant*. The non-perfect DP tradeoff equivalent of $\mathtt{D}_{\text{msc}} = \frac{\vec{m}_0 t^2}{\mathsf{N}}$ will also be used.

Note that Hellman's original matrix stopping rule was $\frac{mt^2}{\mathsf{N}} = 1$ and that the matrix stopping rule used in this paper is equivalent to requiring $\bar{\mathtt{D}}_{\text{msc}} \approx 1$. 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 Perfect rainbow tradeoff

Let us now make the exact variant of the rainbow tradeoff that is treated in this paper more explicit. 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 perfect 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. Unlike the classical Hellman or DP matrices, the rainbow matrix structure quickly becomes inefficient in covering more answers as its dimensions approach something satisfying $mt \approx \mathsf{N}$. 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 $rd_{i,k} : \mathcal{N} \to \mathcal{N}$ $(i = 1, \ldots, \ell, k = 0, \ldots, t-1)$. 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. To complete the parameter setup, another positive integer m_0, that needs to be chosen in a manner to be described below, is fixed.

Algorithm 3: Pre-computation phase of the perfect rainbow tradeoff

```
for i = 1, ..., ℓ do
    RT_i ← ∅;
    for j = 1, ..., m_0 do
        Choose sp^i_j ∈ 𝒩;                                   // starting point
        tp ← sp^i_j;
        for k = 0, ..., t − 1 do                  // generate pre-computation chain
            tp ← F_{i,k}(tp);
        end
        ep^i_j ← tp;                                          // ending point
        Append (sp^i_j, ep^i_j) to RT_i;
    end
    Sort RT_i according to the ep's;                  // non-perfect rainbow table
    R̄T_i ← ∅;
    for each group of (sp, ep)'s in RT_i with a common ep do   // remove ep collisions
        Append any one pair to R̄T_i;
    end
    Record R̄T_i to disk;                                  // perfect rainbow table
end
```

The rudimentary forms of the pre-computation and online phases of the perfect rainbow tradeoff [21] are given by Algorithms 3 and 4, respectively. Further details of the algorithms will be explained below.

The integer m_0 of Algorithm 3 must be set so that each table $\bar{\mathrm{RT}}_i$ is expected to contain m entries. The appropriate value for m_0 is revealed later in this paper by Lemma 9. 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.

The tables $\bar{\mathrm{RT}}_i$ and RT_i produced by Algorithm 3 are called the i-th *perfect* and *non-perfect rainbow tables*, 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 3 in this paper.

Each series of $t+1$ elements spanning from a sp^i_j to the corresponding ep^i_j is a *pre-computation rainbow chain*, and the first part of Algorithm 4 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

Algorithm 4: Online phase of the perfect rainbow tradeoff

```
for s = t − 1, ..., 0 do
    for i = 1, ..., ℓ do
        op ← rd_{i,s}(h) = F_{i,s}(p);
        for k = s + 1, ..., t − 1 do          // generate length-(t − s) online chain
            op ← F_{i,k}(op);
        end
        Search for op among the ep's of R̄T_i;
        if op == ep^i_j then
            tp ← sp^i_j;
            for k = 0, ..., s − 1 do          // regenerate pre-comp chain
                tp ← F_{i,k}(tp);
            end
            if F(tp) == h then                // answer found
                return tp as answer and terminate;
            end
        end
    end
end
```

rainbow chain of length $(t-s)$ for the i-th perfect 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 $\bar{\mathtt{RT}}_i$ 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 $\mathtt{RT}_i$ is the *non-perfect rainbow matrix* $\mathtt{RM}_i$. 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 [21] process of Algorithm 3 does not specify for any specific method to be used in selecting the chain to be retained in the perfect matrix.

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 [2,6,7] are used, so that only $\log m_0$ bits are needed to record each starting point. Each ending point is truncated to a certain length [4,6] to be discussed later in this paper. Finally, the index file technique [4,6] is also applied, so that about $\log m$ further bits per each truncated ending point can be removed with no loss of information.

Those not familiar with the DP and rainbow tradeoffs should note that there are big differences between Algorithms 2 and 4. 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 treats the pre-computation tables in a serial manner, but the rainbow online phase treats the small number of tables in a round-robin fashion. During our analysis, we make the further simplification that all ℓ tables are processed in parallel [21], 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 $\bar{\mathtt{R}}_{\mathrm{msc}} = \frac{mt}{\mathsf{N}}$ will be used for the perfect rainbow tradeoff and this will be refer to as *matrix stopping constant*. The non-perfect rainbow tradeoff analog of $\mathtt{R}_{\mathrm{msc}} = \frac{m_0 t}{\mathsf{N}}$ will also be used.

2.1.3 Checkpoints

The checkpoint technique, introduced in [2] for applications to the rainbow tradeoff, allows for some of the false alarms to be filtered out before the regeneration of the pre-computation chain. This reduces the time complexity of the online phase in exchange for a small increase in storage size. A detailed analysis of the effect a single checkpoint has on the online time complexity was given by [13] and the case of multiple checkpoints was similarly discussed by [19,30].

The checkpoint technique can trivially be extended to the classical Hellman tradeoff and this was analyzed in [13]. On the other hand, for now, there seems to be no literature discussing the application of the checkpoint technique on the DP tradeoff. There are multiple difficulties arising from the variable chain lengths, but we believe it should still be possible to achieve at least some reduction in online time through an application of the checkpoint technique on the DP tradeoff.

Although the checkpoint technique is a meaningful addition to the tradeoff methods, we will not be treating the checkpoint technique in this paper. The practical reason is that its consideration is bound to make our analysis and comparison that are already complicated even more complicated, but there are several other reasons. First of all, the use of checkpoints is tightly linked to the practical requirement for each table entry to be placed at byte-boundaries, and this often makes the theoretical treatment of the associated increase in storage size meaningless. Second, existing analysis [13] has shown that the positive effect of checkpoints is somewhat limited, and we can expect the differences in their effects on different tradeoff methods to be even smaller. Third, while we can expect the checkpoints to be more effective on the rainbow tradeoff than on the DP tradeoff, our later conclusions will roughly be that the rainbow tradeoff is better than the DP tradeoff, even without the use of checkpoints.

In summary, consideration of checkpoints adds much more complexity to the analysis and comparison of the tradeoff algorithms, but the effects of checkpoints are extremely unlikely to change our conclusions, while the practical use of checkpoints will be detached from its theoretical analysis.

2.1.4 Other conventions and comments

The reader is probably aware by now that we are using the symbols $\bar{\mathtt{D}}$, $\mathtt{D}$, $\bar{\mathtt{R}}$, and $\mathtt{R}$, when we wish to denote some value that relates to the perfect DP, non-perfect DP, perfect rainbow, and non-perfect rainbow tradeoffs, respectively.

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 paper without any explanation. A more precise statement of this approximation may be found in [14, Appendix A]. Infinite sums are also frequently approximated by appropriate definite integrals throughout this paper. 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 benefit of using perfect tables outweighs its drawback. Providing information that can be used to settle this question is one of the objectives of this paper. 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.

2.2 Existing analyses of the perfect DP tradeoff

The book [9, p. 100] gives credit to Rivest for first suggesting to apply the DP technique to the classical Hellman tradeoff, but no corresponding formal article was published. The first analysis of the DP tradeoff that attempts to take the non-uniform chain lengths of the DP matrix into account was given by [7,8]. There, credit is given to the unpublished work [22] for also having studied the DP tradeoff independently.

Many interesting variables were introduced by [7,8] while analyzing the perfect DP tradeoff. The first of these is the expected number of chains α after removal of chain merges. The average of DP chain lengths β_0 and $\bar{\beta}$, before and after removal of chain merges, respectively, were also introduced ([7,8] writes $\bar{\beta}$ as β). Note that the variable α is equal to the parameter m used in this paper, but the work [7,8] treated the number of pre-computation chains to be computed before collision removal as a given preset parameter and treated α as a function of the initial chain count. The success probability and online time estimates for the perfect DP tradeoff were given as equations involving α and $\bar{\beta}$. They also stated certain relations satisfied by α, β_0, $\bar{\beta}$, and some other variables. However, they were unable to derive formulas for computing α and $\bar{\beta}$ from the externally provided parameters. Furthermore, as was pointed out by [25], some of their arguments treated the merges of pre-computation chains inadequately and were problematic.

The subsequent work [25] gave a more advanced analysis of the perfect DP tradeoff. They started by computing β_0 for the case when the chain length bounds $\check{t}$ and $\hat{t}$ are both non-trivially enforced ([25] writes β_0 as β). Then the number of distinct nodes expected in a perfect DP matrix was expressed using the variable β_0. Because $\check{t}$ and $\hat{t}$ were taken into consideration while computing the node count, the number of DP chains of any specified length range appearing in a perfect DP matrix could be extracted from the node counts by focusing on sub-matrices of the total DP matrix. The obtained information on the chain length distribution was then used in an ad hoc manner to compute $\bar{\beta}$ ([25] writes $\bar{\beta}$ as β_{mod}). Finally, the distinct ending point count α was easily expressed as a function of the perfect matrix node count and $\bar{\beta}$.

Note that α and the node count for a perfect DP matrix are directly connected to the storage complexity and the success rate of the tradeoff algorithm, respectively. The paper

also provides a simple argument concerning the pre-computation cost and an upper bound on the time complexity of the online phase.

The analysis of the perfect DP tradeoff given by [25] may seem rather complete, except that the effects of false alarms were disregarded during the time complexity analysis. Since we are also claiming to have done the same analysis, a comparison of results is given in Sect. 3.4. Our observation is that the results of [25] are only valid as first approximations, and that these are too rough for the purpose of this paper.

The later work [2] also discussed the perfect DP tradeoff, but they only considered the special case when the DP matrix consists of the maximum number of DP chains that may be collected for a specified DP probability. However, during their analysis, they oddly assumed that the starting points for these chains are DPs. In any case, their result concerning the success probability requires knowledge of the average chain length associated with the maximal perfect DP matrix, but they were unable to provide this value except through experiments. Furthermore, since increasing the number of non-merging rows reduces the average chain length and possibly even the search space coverage, it is unclear if maximal perfect DP tables can be associated with being optimal in any sense.

The final work we mention is [23]. There, a lot of effort was placed in deriving a formula for β_0, but their end result is almost identical to what may be found in [25] ([23] writes β_0 as β). The formulas of [23] and [25] for β_0 will exhibit noticeable differences only when $\hat{t}$ is close to N, which is unrealistically large. After reobtaining β_0, they derived a formula for α that depends on β_0, but the argument was very terse and their logics were not clear. Finally, the two variables α and β_0 were combined to give the success probability of the tradeoff algorithm, but they seem to have confused the concepts of β_0 and $\bar{\beta}$ at this point.

2.3 Existing analyses of the perfect rainbow tradeoff

The introduction of the rainbow tradeoff [21] was accompanied with a rudimentary analysis, which included the worst case online time complexity. The worst case refers to when the online phase algorithm processes all the pre-computation tables without returning the correct answer. However, the effects of false alarms were not accounted for in this worst case complexity claim. They compared the worst case complexity against the similarly rough worst case complexity of the DP tradeoff and claimed that the rainbow tradeoff was more efficient by a factor of two. This was then combined with heuristic arguments, mainly concerning false alarms, to support the claim of even greater advantage. Most of their arguments referred to the non-perfect rainbow tradeoff and the perfect table version made an appearance only at the end of the paper, but the complexity analyses provided were rough enough to be applicable to both versions.

A more refined analysis of the perfect rainbow tradeoff appeared in [2]. It treated the expected online time complexity, rather than the worst case complexity, and took the effects of false alarms into account. Their stated complexity results hold true only in the case of perfect rainbow tables that contain the maximum number of entries possible, but a large part of these results and their proofs can be adjusted to hold true for more general perfect rainbow tables.

The expected online time complexity of the perfect rainbow tradeoff that does not ignore false alarms was also given by [13]. There the complexity results for the general perfect rainbow tables were stated as closed-form formulas. These are easier to use and manipulate than the formulas of [2], which were given as certain double summations that further involved iterative computations if the general perfect rainbow tables were to be considered. However, the results of [2] and [13] should agree accurately when numerically evaluated on any specific

set of reasonable parameters.[2] Our theoretical developments concerning the perfect rainbow tradeoff will rely heavily on these results.

Concerning the success rate of the rainbow tradeoff, note that this is trivial to write down for the perfect table version [21]. A formula for the success rate of even the non-perfect rainbow tradeoff already appeared in [21]. However, iterative computations were required to evaluate the formula on any specific parameter set. A simple closed-form formula that can replace this iterative part, for the special case of N starting points, was presented in [2], while studying the success rate of the maximal perfect rainbow tradeoff. The closed-form formula was slightly modified in [13] to work for any non-perfect rainbow table and was used to study the online complexities of the rainbow tradeoff. The success rate of the non-perfect rainbow tradeoff is not used directly in this work, but plays a crucial role in studying the behavior of false alarms in the perfect rainbow tradeoff, and the current work relies on previous results [2,13] that have worked out these details.

Let us mention one more issue that is not necessarily specific to the perfect rainbow tradeoff, but is closely related to this work. The work [4] claimed that each entry in the pre-computation table for the DP tradeoff can be represented by half the number of bits required for the rainbow tradeoff, but their explanation was rather brief. They followed this claim with a short argument stating that, if the effects of false alarms were to be ignored, one must conclude that the DP tradeoff is twice as efficient as the rainbow tradeoff. An attempt to refute this was made by [2], which maintained that the claim of [4] concerning the required storage bits per table entry was incorrect. With neither [4] nor [2] providing any details, the work [14] clarified that, in the case of non-perfect tradeoffs, the storage requirement comparison of [4] was correct, but that the rainbow tradeoff may still be seen as being advantageous over the DP tradeoff in typical environments. However, the case of the perfect tradeoffs was left untreated.

3 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 accurate description of the DP tradeoff that is being treated in this work was given in Sect. 2.1.1. As will be explained in Sect. 3.4, the results obtained here differ from those that were presented by existing works.

3.1 Online efficiency

This is the most complicated part of this paper. 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 and forming trees. Let us use $\overleftarrow{m}_k$ to denote the number of distinct points expected in its column that is k iterations away from the ending points. In particular, $\overleftarrow{m}_0$ denotes the number of distinct ending points and this is also the number of independent or non-overlapping rows and trees of the non-perfect DP matrix.

[2] The analyses of [2] and [13] both extend further to the use of the checkpoint technique [2] on the perfect rainbow tradeoff, where the two are not in agreement.

Lemma 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 Let us be given a set of $\overleftarrow{m}_1$ points, which are known to be a single F-iteration away from the DPs. We wish to compute the expected size of its F-image. 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 Sect. 2.1.4 that any approximation of $1 + O(\frac{1}{t})$ order multiplicative factor would be ignored and written as an equality. □

More generally, it is possible to show $\overleftarrow{m}_{i+1} \approx \overleftarrow{m}_i$, but it would be unwise to iteratively combine these approximations too many times to conclude $\overleftarrow{m}_j \approx \overleftarrow{m}_i$, for every j and i. In fact, it is easy to argue as in [16] that

$$\overleftarrow{m}_k = |\mathtt{DM}|\left(1 - \frac{1}{t}\right)^{k-1}\frac{1}{t}, \tag{2}$$

for $k \geq 1$, so that $\overleftarrow{m}_j = \left(1 - \frac{1}{t}\right)^{j-i}\overleftarrow{m}_i$. Here, the $|\mathtt{DM}|$ denotes the number of distinct points expected in a non-perfect DP matrix. 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.

In passing, we caution the reader that one must be aware of the possibility of erring when extending (2) to the $k = 0$ case and writing

$$\overleftarrow{m}_0 = |\mathtt{DM}|\left(1 - \frac{1}{t}\right)^{-1}\frac{1}{t} \approx |\mathtt{DM}|\frac{1}{t}\left(1 + \frac{1}{t}\right), \qquad \text{(problematic!)} \tag{3}$$

since one can infer from the proof of Lemma 1 that

$$\overleftarrow{m}_0 \approx \overleftarrow{m}_1\left\{1 - \frac{(\overleftarrow{m}_1 - 1)\,t}{2\mathsf{N}}\right\} \approx |\mathtt{DM}|\frac{1}{t}\left(1 - \frac{\bar{\mathrm{D}}_{\mathrm{msc}}}{2t}\right), \tag{4}$$

with the opposite sign in the second term, is a much more reasonable and accurate approximation. These two expressions for $\overleftarrow{m}_0$ are certainly close to each other and also to $\overleftarrow{m}_1$, so that the use of (3) could be acceptable under many circumstances, but it would be inappropriate to claim (3) by itself.

The core information missing from (2) is also already available. It is known [14] that a single non-perfect DP matrix created with $\vec{m}_0$ starting points is expected to contain

$$|\mathtt{DM}| = \frac{2\vec{m}_0\, t}{1+\sqrt{1+2\mathtt{D}_{\mathrm{msc}}}} \tag{5}$$

distinct points, where $\mathtt{D}_{\mathrm{msc}} = \frac{\vec{m}_0 t^2}{\mathsf{N}}$ is the matrix stopping constant for the 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 we have gathered so far can be used to relate the number of starting points to the number of distinct ending points.

Lemma 2 *A non-perfect DP matrix created with $\vec{m}_0$ 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 Lemma 1 states that the number of distinct ending points $\overleftarrow{m}_0$ may be approximated by $\overleftarrow{m}_1$. In fact, since we are ignoring approximations of $1+O\left(\frac{1}{t}\right)$ order multiplicative factors, we may even write $\overleftarrow{m}_0 = \overleftarrow{m}_1$. Recalling from (2) that $\overleftarrow{m}_1 = |\mathtt{DM}|\frac{1}{t}$ and combining this with (5), 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, it suffices to solve for $\vec{m}_0$ from the relation

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

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

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

and again into the form

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

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

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

which is the second claim. □

Note that the first sentence of this lemma gives a simple formula for α, the number of chains remaining after the removal of merges, discussed in Sect. 2.2, which many previous works had attempted to find.

For the remainder of this section,

$$\vec{m}_0 = \left(1 + \frac{\bar{\mathtt{D}}_{\mathrm{msc}}}{2}\right) m \tag{6}$$

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 $\vec{m}_0$ 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 $\mathtt{D}_{\mathrm{msc}} = \frac{\vec{m}_0 t^2}{\mathsf{N}}$ and $\bar{\mathtt{D}}_{\mathrm{msc}} = \frac{mt^2}{\mathsf{N}}$, we can rewrite the above as

$$\mathtt{D}_{\mathrm{msc}} = \left(1 + \frac{\bar{\mathtt{D}}_{\mathrm{msc}}}{2}\right) \bar{\mathtt{D}}_{\mathrm{msc}}. \tag{7}$$

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

$$\bar{\mathtt{D}}_{\mathrm{msc}} = \sqrt{1 + 2\mathtt{D}_{\mathrm{msc}}} - 1. \tag{8}$$

This can be used to convert any formula given in terms of $\bar{\mathtt{D}}_{\mathrm{msc}}$ into one given in terms of $\mathtt{D}_{\mathrm{msc}}$.

One consequence of creating a DP matrix from $\vec{m}_0$ starting points, where $\vec{m}_0$ is as given by (6), is the interesting formula

$$|\mathtt{DM}| = mt, \tag{9}$$

which is evident from the first equation in the proof to Lemma 2. 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 (6), is expected to cover mt distinct points.

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 $\bar{\mathtt{D}}_{\mathrm{pc}} = \frac{\vec{m}_0 t \ell}{\mathsf{N}}$, so that the cost of pre-computation becomes $\bar{\mathtt{D}}_{\mathrm{pc}}\mathsf{N}$. The following statement is a direct consequence of Lemma 2 or (6).

Proposition 1 *The pre-computation coefficient of the perfect DP tradeoff is*

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

By the definition of the coverage rate, which was given at the end of Sect. 2.1.1, a single perfect DP matrix has probability $\frac{mt\,\bar{\mathtt{D}}_{\mathrm{cr}}}{\mathsf{N}}$ of containing the correct answer to the given inversion problem. Thus, the success probability of the complete perfect DP tradeoff may be stated as

$$\bar{\mathtt{D}}_{\mathrm{ps}} = 1 - \left(1 - \frac{mt\,\bar{\mathtt{D}}_{\mathrm{cr}}}{\mathsf{N}}\right)^{\ell} = 1 - \exp\left(-\frac{mt\ell}{\mathsf{N}}\bar{\mathtt{D}}_{\mathrm{cr}}\right), \tag{10}$$

where we are relying on the approximation stated in Sect. 2.1.4 for the second equality, and we can combine this with Proposition 1 to claim the following.

Proposition 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 succeeded in obtaining expressions for $\bar{\mathrm{D}}_{\mathrm{pc}}$ and $\bar{\mathrm{D}}_{\mathrm{ps}}$ that do not involve $\vec{m}_0$. Our next short term objective is to obtain such an expression for $\bar{\mathrm{D}}_{\mathrm{cr}}$. Some technical lemmas need to be prepared first.

Given a function $F : \mathscr{N} \to \mathscr{N}$ and a nonnegative integer k, we define $\mathscr{D}_k(F)$ or $\mathscr{D}_k$ to be the set of elements of $\mathscr{N}$ 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 $\{\mathscr{D}_k(F)\}_{k=0}^{\infty}$ is a partition of $\mathscr{N}$, and that we can expect the sizes of these subsets to be

$$|\mathscr{D}_k| = \mathsf{N}\left(1-\frac{1}{t}\right)^k \frac{1}{t}, \tag{11}$$

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 $t \ll \sqrt{N}$ and the rho length of a random walk initiated from a random point is expected to be of $\sqrt{N}$ order.

Lemma 3 *Let $F : \mathscr{N} \to \mathscr{N}$ be chosen uniformly at random from the set of all functions acting on $\mathscr{N}$ 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 $i = 1, \ldots, k$.

Proof A trivial generalizing of the argument we saw in the proof of Lemma 1 shows that, 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 is expected to be

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

assuming $|\mathscr{C}| = O(|\mathscr{B}|)$. The claim is now a direct consequence of the set sizes given by (11). Note that, since $|\mathscr{D}_j| \leq |\mathscr{D}_{j-1}|$, we need not worry about the $|\mathscr{C}| = O(|\mathscr{B}|)$ condition. A more detailed proof is provided in Appendix 3, for those interested in the subtleties hidden behind this short argument. □

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

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

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

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 as

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

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

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

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

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

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

The previous two lemmas were prepared to support the next lemma, which gives the probability for a single chain to merge into a set of chains. This information will be used to study the inner workings of how a perfect DP table is formed from a non-perfect DP table.

Lemma 5 *Let $F : \mathcal{N} \to \mathcal{N}$ be a random function and let $D \subset \mathscr{D}_k(F)$, for some k. When $|D| = O(m)$, the probability for a random point $x \in \mathscr{D}_k(F)$ to satisfy $F^k(x) \notin F^k(D)$ is*

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

Proof Since the starting point itself and each subsequent iterations of the random function must land outside the iterated image sets, the probability in question is

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

Here, the first equality is based on (11). By applying Lemma 3 to the product of k terms, we can write

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

Since we are given the condition $|D|=O(m)$, we can apply Lemma 4, or the last equation in its proof, and compute the sum inside the exponential function as

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

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

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

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

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

The $\left(1-\frac{1}{t}\right)$ 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 prepared, we can finally present something of more direct practical value.

Proposition 3 *The coverage rate of a perfect DP matrix is*

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

Proof Consider a pre-computed non-perfect DP matrix and the process of removing chains from it to obtain a perfect DP matrix. A chain survives through the collision removal process if and only if it does not merge into another chain that is longer than (or equal to) its length. Hence, according to Lemma 5, the probability for a chain of length k in a non-perfect DP table to remain in the perfect table is

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

This figure is a slight underestimate since the merges among chains of the same length were reflected too many times, but such merges are rare and will not cause noticeable inaccuracy.

Since the 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 the perfect table contains no overlapping of points, the number of distinct points in the perfect DP table is

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

This formula does not count the ending points and only includes the points that were used as inputs to the iterating function during the DP table computation.

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

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

where we have used (2) and (9) to remove the $\overleftarrow{m}_k$ term. After ignoring the insignificant $(1-\frac{1}{t})^{-1}$ terms, we rewrite the above as

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

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

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

It now suffices to recall Lemma 2 or (6) to arrive at the claimed formula. □

Let us briefly digress and discuss the average chain length $\bar{\beta}$ of a perfect DP matrix that was introduced in Sect. 2.2. By definition, it is the number of points in a perfect DP matrix divided by the number of its ending points, and the above lemma allows us to write it as

$$\bar{\beta}=\frac{|\bar{\mathrm{DM}}|}{m}=\frac{mt\bar{\mathrm{D}}_{\mathrm{cr}}}{m}=t\,\frac{2}{\bar{\mathrm{D}}_{\mathrm{msc}}}\ln\left(1+\frac{\bar{\mathrm{D}}_{\mathrm{msc}}}{2}\right).\tag{12}$$

Should it be required, we can use (8) to rewrite this in terms of the parameters $\vec{m}_0$ and t as

$$\bar{\beta}=t\,\frac{1+\sqrt{1+2\mathrm{D}_{\mathrm{msc}}}}{\mathrm{D}_{\mathrm{msc}}}\ln\left(\frac{1+\sqrt{1+2\mathrm{D}_{\mathrm{msc}}}}{2}\right),\tag{13}$$

where $\mathrm{D}_{\mathrm{msc}}=\frac{\vec{m}_0t^2}{\mathrm{N}}$. It is easy to check that this $\bar{\beta}$ value is always smaller than the average chain length $\beta_0=t$ 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.

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 Sect. 3.3 that, for a wide range of parameters m and t, which covers all parameter combinations of interest, the value computed through the formula

$$t \times \frac{1 + 0.577\,\bar{\mathrm{D}}_{\mathrm{msc}}}{1 + 0.451\,\bar{\mathrm{D}}_{\mathrm{msc}}} \tag{14}$$

agrees accurately with the experimentally obtained average number of one-way function iterations required for this partial pre-computation chain regeneration.

Let us clarify that we are not claiming formula (14) to be *correct* in any theoretical sense. In fact, we know that the seemingly very different formula

$$t \left(1 + 0.340468 \left\{1 - \frac{1.32798}{\bar{\mathrm{D}}_{\mathrm{msc}}} \ln\left(1 + \frac{\bar{\mathrm{D}}_{\mathrm{msc}}}{1.32798}\right)\right\}\right) \tag{15}$$

works equally well for parameters of interest. Our only claim here is that formula (14) predicts the average cost of resolving each alarm with accuracy that is more than sufficient for most practical purposes.

Proposition 4 *The online processing of a single perfect DP table is expected to require*

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

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

Proof As the work factor (14) is already available, it only remains to find the probability of encountering an alarm.

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 (9) states the number of elements contained in $\mathtt{DM}$ as mt, the probability of merge can be stated as

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

The claimed expected cost of dealing with a possible alarm is the product of this probability and the work factor (14). □

Having obtained the cost of dealing with alarms, the online complexities of the perfect DP tradeoff can be gathered in a single tradeoff curve.

Theorem 1 *The time memory tradeoff curve for the perfect DP tradeoff is* $T M^2 = \bar{\mathrm{D}}_{\mathrm{tc}} \mathsf{N}^2$*, where the tradeoff coefficient is given by*

$$\bar{\mathrm{D}}_{\mathrm{tc}} = \left(1 + \frac{1 + 0.577\,\bar{\mathrm{D}}_{\mathrm{msc}}}{1 + 0.451\,\bar{\mathrm{D}}_{\mathrm{msc}}} \frac{\bar{\mathrm{D}}_{\mathrm{msc}}}{1 + \bar{\mathrm{D}}_{\mathrm{msc}}}\right) \frac{\bar{\mathrm{D}}_{\mathrm{ps}} \left\{\ln(1 - \bar{\mathrm{D}}_{\mathrm{ps}})\right\}^2}{\bar{\mathrm{D}}_{\mathrm{msc}}\, \bar{\mathrm{D}}_{\mathrm{cr}}^3}.$$

Proof Since a single perfect DP matrix has probability $\frac{mt\,\bar{\mathrm{D}}_{\mathrm{cr}}}{\mathsf{N}}$ of containing the correct answer to a given inversion problem, the probability for the i-th DP table to be processed during the online phase executed for a single inversion target is $\left(1 - \frac{mt\bar{\mathrm{D}}_{\mathrm{cr}}}{\mathsf{N}}\right)^{i-1}$. The online processing of each table is expected to require t iterations of the one-way function for the online chain generation and the expected number of iterations required to deal with the alarm that could occur is given by Proposition 4. Hence, the number of one-way function iterations expected during the online phase is

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

where the final equality relies on (10) or Proposition 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:

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

The second equality here is obtained through another application of (10). □

Let us clarify that both Proposition 4 and Theorem 1 depend on the empirical result (14). 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.

3.2 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 (6) 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.

To deal with the ending point storage, the effects of truncating ending points before recording them to the pre-computation tables need to be discussed, and this will require us to take a brief digression. Let us state that a function $\varphi : \mathscr{A} \to \mathscr{B}$ is *pre-image uniform*, if the pre-image set $\varphi^{-1}(b) \subset \mathscr{A}$ contains the same number of elements for every $b \in \mathscr{B}$. A pre-image uniform function must clearly be surjective.

Lemma 6 *Let $\varphi : \mathscr{A} \to \mathscr{B}$ be a pre-image uniform function and let $|\mathscr{A}| \gg |\mathscr{B}|$. When $x = O(|\mathscr{B}|)$ distinct elements of $\mathscr{A}$ are chosen at random, the expected size of the set formed by their φ-images is $|\mathscr{B}|\{1 - \exp(-\frac{x}{|\mathscr{B}|})\}$. Conversely, given $y \leq (1 - e^{-5})|\mathscr{B}|$, one must choose $|\mathscr{B}| \ln(\frac{|\mathscr{B}|}{|\mathscr{B}| - y})$ distinct elements of $\mathscr{A}$ at random in order for y to be the expected size of the set formed by their φ-images.*

Proof It is clear that if one makes x independent random selections from the co-domain $\mathscr{B}$, then the number of distinct elements observed is expected to be

$$|\mathscr{B}| \left\{ 1 - \left(1 - \frac{1}{|\mathscr{B}|} \right)^x \right\}.$$

Furthermore, as was stated in Sect. 2.1.4, this can be approximated by the value given in the first claim, under the condition $x = O(|\mathscr{B}|)$. Hence, to arrive at the first claim, it suffices to show that selecting elements of $\mathscr{B}$ by first randomly selecting x distinct elements of $\mathscr{A}$ and then applying φ to them is very close to making x independent random selections from $\mathscr{B}$.

To this end, it should first be noted that the pre-image uniform property of φ ensures that the random selection of a single element from $\mathscr{A}$ will be translated through an application of φ to the random selection of an element from $\mathscr{B}$. Thus, our focus will be on how much effect the condition of x inputs to φ being distinct, rather than each being selected independently, has on this translation of the uniform random distribution on $\mathscr{A}$ to the uniform random distribution on $\mathscr{B}$.

Let us consider the partition on $\mathscr{A}$ into $|\mathscr{B}|$-many equally sized cells that is naturally given by φ. In other words, we define two element of $\mathscr{A}$ to be equivalent if φ maps them to the same element of $\mathscr{B}$. Then, the selection of a $\mathscr{B}$-element carried out by applying φ to a selected $\mathscr{A}$-element could be seen as selecting an $\mathscr{A}$-element and simply taking note of the cell it belongs to. Next, let us view the selection of x distinct $\mathscr{A}$-elements as a series of x incremental selections, randomly made each time among those $\mathscr{A}$-elements that have not yet been selected.

Now, the condition $|\mathscr{A}| \gg |\mathscr{B}|$ implies that the (uniform) cell size $\frac{|\mathscr{A}|}{|\mathscr{B}|}$ for the partition on $\mathscr{A}$ will be large. Hence, the previous selection(s) of a cell through one or two of its elements will have very little effect on the possibility of that cell being selected again. Furthermore, since $x = O(|\mathscr{B}|)$, cells that have been chosen more than a few times will be rare, even towards the end of all x selections, if the selections are made randomly. Thus, the x incremental exclusive random selections on $\mathscr{A}$ and applications of φ to them is very close to a series of x independent random selections from $\mathscr{B}$.

To arrive at the second claim, it now suffices to solve for x in the relation

$$y = |\mathscr{B}| \left\{ 1 - \exp\left(-\frac{x}{|\mathscr{B}|} \right) \right\}.$$

The somewhat arbitrarily chosen condition $y \leq (1 - e^{-5})|\mathscr{B}|$ is equivalent to $x \leq 5|\mathscr{B}|$ and ensures that the condition $x = O(|\mathscr{B}|)$, required by the first claim, is satisfied. □

We now return to the subject of ending point truncation. 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 $\mathscr{N}$. 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.

The effects of ending point truncation on the perfect DP tradeoff are slightly different from those on the non-perfect DP tradeoff, which were treated in [14]. The truncation may cause two pre-computation chains that do not merge into each other to become indistinguishable at the ending points and cause more chains to be discarded. However, our next lemma shows that these *pseudo-collisions* can mostly be avoided by recording slightly more than $\log m$ bits.

Note that any natural truncation map will be, or will be very close to being, pre-image uniform. In fact, truncation maps that are far from being pre-image uniform should be avoided, as they will lead to more collisions among truncated ending points.

Lemma 7 *Consider a (pre-image uniform) 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 $r\left\{1 - \exp(-\frac{m}{r})\right\}$ distinct truncated points. Conversely, given $m \leq (1 - e^{-5})r$, one must truncate $r \ln\left(\frac{r}{r-m}\right)$ distinct ending point DPs in order for m to be the expected number of distinct truncated points.*

No proof of this lemma will be provided, since this is a direct translation of Lemma 6 into the language of the DP tradeoff.

Let us consider a specific example. When the truncated space is of size $r = 2^5 m$, it suffices to truncate $32m \ln\left(\frac{32}{31}\right) = 1.01596\,m$ DPs in order to obtain m distinct truncated ending points. Combining this information with Proposition 1, one can state that, by recording just $5 + \log m$ bits of each ending point, one can control the extra pre-computation necessitated by the ending point truncation to within approximately 2 %. 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 1, but also inside the $\frac{\bar{\mathrm{D}}_{\mathrm{msc}}}{2}$ term. 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 falsely announcing a match and then regenerating the pre-computation chain to resolve this alarm.

Lemma 8 *Consider a (pre-image uniform) 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*

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

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 through iterations of the random function in the following manner: (1) Random choices for the first i nodes of the online chain, starting from the correct pre-image of the inversion target, must be made among the non-DPs that do not belonging to the non-perfect DP matrix `DM`; (2) The final point must be chosen among DPs that are different from the m ending points; (3) Furthermore, the final point must be chosen so that its truncation 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 points we know the final point not to be, i.e., $\frac{\mathsf{N}}{t} \gg m$, the dependence can be ignored. Thus, the probability we seek is

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

where we have used $\frac{m}{\mathsf{N}}=O(\frac{1}{mt})=o(\frac{1}{t})$ for the approximation and (9) for the final equality. Thus, the probability for the online processing of a perfect DP table to cause a truncation-related alarm, i.e., an alarm that does not involve the online chain merging into the pre-computation matrix, is given by

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

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 pseudo-collision is expected to be the average chain length of the perfect DP matrix, which is given by (12). The cost of resolving alarms that are induced by truncation is

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

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 1 to be

$$\left(1+\frac{1+0.577\,\bar{\mathrm{D}}_{\mathrm{msc}}}{1+0.451\,\bar{\mathrm{D}}_{\mathrm{msc}}}\,\frac{\bar{\mathrm{D}}_{\mathrm{msc}}}{1+\bar{\mathrm{D}}_{\mathrm{msc}}}\right)t. \tag{16}$$

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 $\bar{\mathrm{D}}_{\mathrm{msc}}=1$, the expected numbers of normal iterations and truncation-related iterations become $\left(1+\frac{1.577}{1.451}\frac{1}{2}\right)t=1.54342t$ and $\ln(\frac{3}{2})\frac{m}{r}t=0.405465\frac{m}{r}t$, respectively. For example, with $r=2^5m$, the ending point truncation increases the number of one-way function iterations by a mere $\frac{0.405465\frac{1}{32}t}{1.54342t}\approx 0.82\ \%$. The following can be stated for the general situation.

Proposition 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 $r=2^{\epsilon}m$. 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*

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

additional iterations of the one-way function to complete.

For parameters satisfying $\bar{\mathrm{D}}_{\mathrm{msc}}=1$, the above is $\frac{0.405465}{1.54342}\frac{T}{2^{\epsilon}}=0.262706\frac{T}{2^{\epsilon}}$. 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. This was also the conclusion obtained for the non-perfect DP tradeoff in [14].

3.3 Experimental 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 instantiated with the key to ciphertext mapping, under a randomly fixed plaintext, of the blockcipher 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 2 and Proposition 3 simultaneously. Recall that Lemma 2 related the number of starting points to the number of distinct ending points in a non-perfect DP matrix and that Proposition 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 (6). 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 $\bar{\mathrm{D}}_{\mathrm{cr}}$ 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 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

Table 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	$\bar{\mathrm{D}}_{\mathrm{msc}}$	$\vec{m}_0$ used	Test m	Theoretical $\bar{\mathrm{D}}_{\mathrm{cr}}$	Test $\bar{\mathrm{D}}_{\mathrm{cr}}$
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

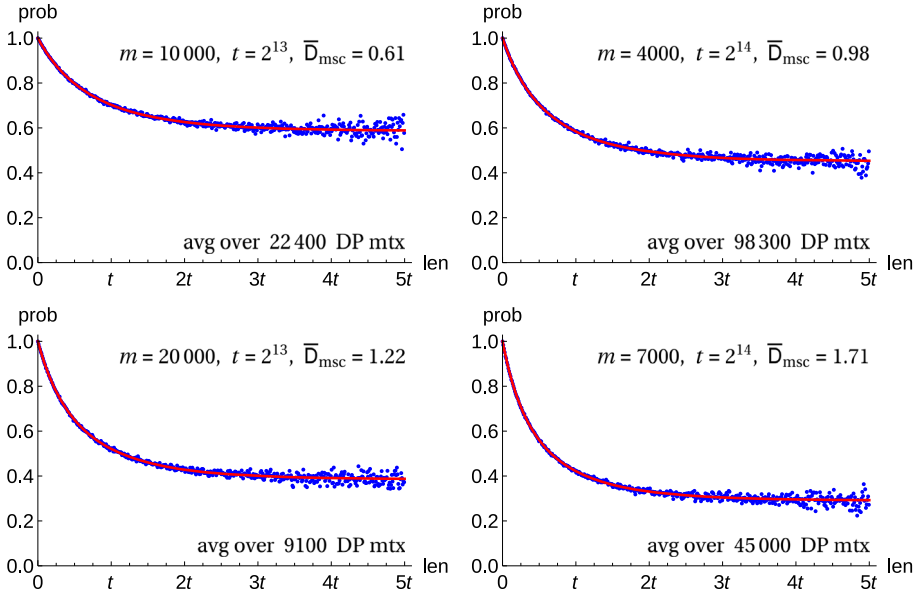

Fig. 1 The probability for DP chains of each length to survive through the treatment of merging chains in a DP matrix. (test: *dots*; theory: *line*; $\mathsf{N} = 2^{40}$; $\hat{t} = 15\,t$)

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 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

$$\left\{1 + \frac{\bar{\mathsf{D}}_{\mathrm{msc}}}{2}\left(1 - e^{-\frac{k}{t}}\right)\right\}^{-2} \tag{17}$$

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 $\hat{t} = 15\,t$. Then, for each $1 \le k < \hat{t}$, 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 Fig. 1. The probability (y-axis) for chain survival through the chain merge removal process is given for each chain length (x-axis). The lines correspond

to our theory, as given by (17), and the dots represent the count ratios obtained through tests. Even though our chain length bound was $\hat{t} = 15\,t$, 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 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.

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 Sect. 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 (14), are summarized in Table 2. We have also plotted the experiment data of Table 2 and the curve given by formula (14) in Fig. 2. The test 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 (14) is very close to the average number of one-way function iterations required per alarm that was obtained experimentally. Also, after viewing Fig. 2, one can be confident that formula (14) will be quite accurate, at least for all parameter choices satisfying $0 < \bar{\mathrm{D}}_{\mathrm{msc}} < 2.3$.

3.4 Comments on a previous analysis of the perfect DP tradeoff

Since there is a large overlap between what the previous work [25] and the current work claim to have obtained, let us compare some of the results from the two works.

The paper [25] provided a method for computing the number of distinct ending DPs expected from a given number of starting points. This value, which is the number of chains remaining after removal of merging chains, was referred to as α in Sect. 2.2. In Table 3, we have copied some of the related data appearing in [25, Table 2] and have appended our corresponding theoretical values, computed with Lemma 2. The parameters associated with this table are $\mathsf{N} = 2^{56}$ and $t = 2^{18}$, and all figures in the table are given as $\log_2$ values. Recall that Lemma 2 is valid for the case when no chain length bounds are set. The data from [25] are for when there is no lower bound on the chain length and the chain length upper bound is set to 2^{30}, which is more than sufficiently large for the $t = 2^{18}$ being used.

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

Parameters				Test	Formula	Test / Formula
$\vec{m}_0$	m	t	$\bar{\mathsf{D}}_{\text{msc}}$	$\frac{1}{t} \times$ #(itr)	$\frac{1}{t} \times$ Eq.(14)	
#(tbl)	#(alarm) / tbl					
3000	2766	131072	0.16882	1.02002	1.01977	1.00025
1280	10000					
231000	209976	16384	0.20025	1.02354	1.02314	1.00039
128	30000					
17000	15230	65536	0.23239	1.02616	1.02650	0.99967
640	10000					
48000	37354	65536	0.56998	1.05758	1.05713	1.00042
128	10000					
12775	9843	131072	0.60077	1.05889	1.05956	0.99937
640	5000					
870000	661405	16384	0.63077	1.06202	1.06187	1.00014
128	20000					
1504000	1013856	16384	0.96689	1.08460	1.08483	0.99978
128	20000					
99000	65884	65536	1.00531	1.08758	1.08715	1.00039
128	10000					
6400	4223	262144	1.03101	1.08824	1.08867	0.99961
1280	10000					
9400	5588	262144	1.36426	1.10606	1.10642	0.99967
1280	10000					
156000	91761	65536	1.40016	1.10806	1.10814	0.99993
128	20000					
2576000	1501280	16384	1.43173	1.10999	1.10962	1.00033
128	20000					
217500	115580	65536	1.76361	1.12370	1.12377	0.99994
128	20000					
3587000	1887750	16384	1.80030	1.12553	1.12519	1.00030
128	20000					
14400	7512	262144	1.83398	1.12610	1.12647	0.99967
1280	10000					
74000	35512	131072	2.16748	1.13818	1.13810	1.00007
128	10000					
4845000	2307050	16384	2.20017	1.13976	1.13915	1.00054
128	20000					
310000	146425	65536	2.23427	1.14000	1.14022	0.99981
128	20000					

One can verify, in every row of the table, that our theory provides estimates that are closer to the test results of [25] than their own theory. Some might dismiss the differences between the two theoretical values as being too small to be of any significance, but one must consider the fact that these are $\log_2$ values. For example, referencing the last row of the table, we can see that the (in)accuracy of Lemma 2 in predicting the test value is by a factor of $\frac{2^{21.6425}}{2^{21.6430}} \approx 0.999653$, but that the theoretical prediction of [25] is further away at a factor of

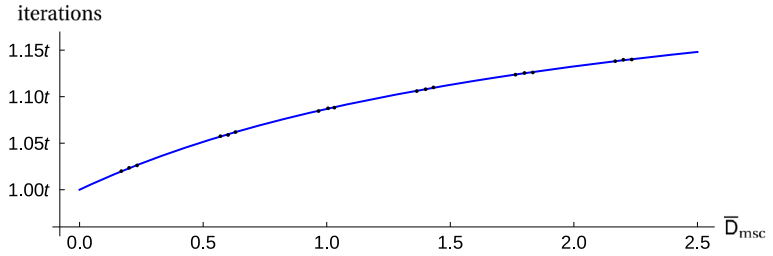

Fig. 2 The number of one-way function iterations required to resolve each alarm, plotted in relation to the $\bar{\mathrm{D}}_{\mathrm{msc}}$ value for the parameters that were used (test: *dots*; theory: *line*; $\mathsf{N} = 2^{48}$; $\hat{t} = 15\,t$)

Table 3 Number of DP chains before and after removal of chain merges ($\mathsf{N} = 2^{56}$, $t = 2^{18}$). All values are given in $\log_2$ scale

# of starting points	# of ending points		
	Test of [25]	Theory of [25]	Lemma 2
20	19.5497	19.4712	19.5500
21	20.3048	20.1357	20.3058
22	20.9990	20.6866	21.0000
23	21.6425	21.1357	21.6430

$\frac{2^{21.6425}}{2^{21.1357}} \approx 1.42$. The error factor of 1.42 in the estimated number of ending points is directly reflected in the pre-computation table size estimate and presents itself as a $1.42^2 \approx 2.02$ error factor in the estimate for the tradeoff coefficient. Hence, for the purpose of comparing the online efficiencies of tradeoff algorithms, which usually differ only by factors of such small (but still practically significant) magnitude, the theory of [25] was not accurate enough.

Recall from Sect. 2.2 that the expression for α given in [25] involved the average chain length $\bar{\beta}$ and that their derivation of $\bar{\beta}$ involved an ad hoc procedure. So, let us consider the possibility that this ad hoc procedure was the cause of the inaccuracy we have witnessed in Table 3 and that their arguments were correct up to that point. For this purpose, we focus on one major intermediate result of [25] which they referred to as the storage function $s(j)$.

The exact definition of $s(j)$ is slightly complicated and will not be explained here in full, but when no bounds are placed on the chain lengths, function $s(j)$ is suppose to give the expected number of distinct nodes in a perfect DP matrix that was created with j starting points. The formula they gave was

$$s(j) = 2^k \left\{ 1 - \left(\frac{2^k}{-\beta j + \beta^2 j + K} \right)^{\frac{1}{\beta - 1}} \right\} \quad \text{with} \quad K = 2^k \left(1 - \frac{s(0)}{2^k} \right)^{1-\beta}, \tag{18}$$

where the β they used was the average chain length before the removal of chain merges and 2^k was the size of the search space. The presence of $s(0)$ may seem strange, but this is because we have not explained the meaning of $s(j)$ in full, and we may simply take $s(0) = 0$ for this discussion. In the absence of chain length bounds, we may take $\beta = t$ and approximate their formula into

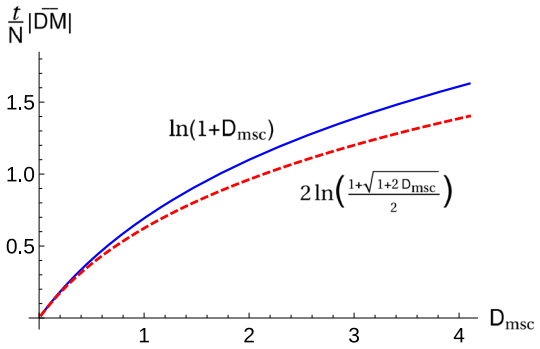

Fig. 3 The value $\frac{t}{N}|\bar{DM}| = \bar{D}_{msc}\bar{D}_{cr}$ as predicted by [25] (*solid*) and the current work (*dashed*)

$$s(j) = N\left\{1 - \left(\frac{N}{t^2 j + N}\right)^{\frac{1}{t}}\right\}, \tag{19}$$

written in the notation of this paper. Hence, according to the theory of [25], if one creates a perfect DP matrix using $\vec{m}_0$ starting points, there will be

$$|\bar{DM}| = s(\vec{m}_0) = N\left\{1 - \left(\frac{1}{1 + D_{msc}}\right)^{\frac{1}{t}}\right\} \tag{20}$$

distinct points in the matrix, where $D_{msc} = \frac{\vec{m}_0 t^2}{N}$. To allow for direct comparison with our result, we rewrite this in the form

$$\frac{t}{N}|\bar{DM}| = t\left\{1 - \left(\frac{1}{1 + D_{msc}}\right)^{\frac{1}{t}}\right\} \approx \ln(1 + D_{msc}). \tag{21}$$

The approximation here is accurate for all sufficiently large t. For example, at $D_{msc} = 1$ the right-hand value is $\ln(1 + D_{msc}) = 1.09861$, and even at the moderately sized $t = 2^{10}$, the formula in the middle involving t evaluates to 1.09802.

Our corresponding result is contained in Proposition 3. Using (8), we can claim

$$\frac{t}{N}|\bar{DM}| = \bar{D}_{msc}\bar{D}_{cr} = 2\ln\left(1 + \frac{\bar{D}_{msc}}{2}\right) = 2\ln\left(\frac{1 + \sqrt{1 + 2D_{msc}}}{2}\right), \tag{22}$$

which is clearly different from (21).

The numeric values given by the two formulas (21) and (22) are compared in Fig. 3. The two are visibly different, except when D_{msc} is very small, in which case chain merges are rare and both theories reduce to something trivial. For example, even at $D_{msc} = 1$, the two theories give clearly different values of 0.69315 and 0.62381. Since the testing of Sect. 3.3 has shown our estimate of coverage rate to be much more accurate than the general order of these differences, the storage function $s(j)$, as given by (19), must not be as accurate.

Let us explain the source of their inaccuracy. The core logic of [25] rests in their Eq. (22), which we approximate and state as

$$s(j + 1) = s(j) + t\left(1 - \frac{s(j)}{N}\right)^t, \tag{23}$$

for the case when there are no chain length bounds. This equation can be interpreted as follows. One expects to add t new points to the perfect matrix by considering one more pre-computation chain, but this addition should only be done when the chain of t points do not merge with the points within the existing matrix. The t-th powered term on the right is the probability for none of the t additionally generated points to be in the existing matrix of $s(j)$ points. This high-level view of (23) may seem plausible.

The first source of inaccuracy hiding in (23) is the assumption that the new chain added is always of length t. In reality, chains of varying lengths will be created and these all have different probabilities of merging with the existing matrix. The simplified view of [25] could have been justified to a certain degree if the chain lengths were mostly close to t, but the actual distribution of chain lengths is not even centered at the average value t. The authors of [25] seem to have been aware of this problem. In later parts of their paper, they divide the range of possible chain lengths into a small number of segments and treat each length range separately, using different average chain lengths within each segment.

The second source of inaccuracy is in the t-th powered term that was suppose to give the probability for the additional chain not to merge with the existing matrix. Since we are dealing with a DP chain of a predetermined length at this point, a more accurate expression would somehow involve the set sizes $|\mathscr{D}_k|$, that appeared in the main body of this work, in the denominator, rather than N. Furthermore, the numerator $s(j)$ should be replaced with values that are associated in some way with the *non-perfect* matrix created up to that point. This is because the newly created chain is destined to merge with the perfect matrix whenever it merges with a previously generated chain, regardless of whether the chain has been discarded due to merging.

We conclude that while the analyses of [25] were based on plausible arguments, their results are valuable only as first approximations and are not accurate enough for the purpose of this paper, which is to compare the performances of different tradeoff algorithms.

4 Perfect rainbow tradeoff

This section gathers facts concerning the perfect rainbow tradeoff that are required for our later 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 [13,21], and [14] should be able to reproduce the claims of this section.

4.1 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 in an appropriate manner.

Lemma 9 *A non-perfect rainbow matrix created with m_0 starting points is expected to contain $\frac{2m_0}{2+\mathrm{R}_{\mathrm{msc}}}$ distinct ending points, where $\mathrm{R}_{\mathrm{msc}} = \frac{m_0 t}{\mathsf{N}}$. Conversely, given m, one must generate $m_0 = \frac{2}{2-\bar{\mathrm{R}}_{\mathrm{msc}}} m$ chains, where $\bar{\mathrm{R}}_{\mathrm{msc}} = \frac{mt}{\mathsf{N}}$, 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 [2,13] 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 $0 \leq i \leq t$. 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. □

For the remainder of this section,

$$m_0 = \frac{2}{2 - \bar{\mathsf{R}}_{\mathrm{msc}}} m \tag{24}$$

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 3, given the algorithm parameters m and t.

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 [2,21], assuming a fixed t. Substituting $m_0 = \mathsf{N}$ into the second equation appearing in the proof of Lemma 9, we see that $m = m_t = \frac{2\mathsf{N}}{2+t}$. This implies an upper bound

$$\bar{\mathsf{R}}_{\mathrm{msc}} \leq \frac{2t}{t+2} < 2 \tag{25}$$

on the possible range of $\bar{\mathsf{R}}_{\mathrm{msc}}$, with the possibility of $\bar{\mathsf{R}}_{\mathrm{msc}}$ 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 $\bar{\mathsf{R}}_{\mathrm{pc}} = \frac{m_0 t \ell}{\mathsf{N}}$, so that the number of one-way function iterations required for the pre-computation phase becomes $\bar{\mathsf{R}}_{\mathrm{pc}}\mathsf{N}$. The following statement is a direct consequence of Lemma 9.

Proposition 6 *The pre-computation coefficient of the perfect rainbow tradeoff is*

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

The success probability of a perfect rainbow tradeoff may easily be stated [2] as

$$\bar{\mathrm{R}}_{\mathrm{ps}} = 1 - \left(1 - \frac{m}{\mathsf{N}}\right)^{t\ell} = 1 - \exp\left(-\frac{mt\ell}{\mathsf{N}}\right) = 1 - \exp\left(-\bar{\mathrm{R}}_{\mathrm{msc}}\ell\right), \tag{26}$$

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

$$\bar{\mathrm{R}}_{\mathrm{msc}} = -\frac{\ln\left(1 - \bar{\mathrm{R}}_{\mathrm{ps}}\right)}{\ell} \tag{27}$$

one must adhere to, when selecting parameters that achieve a prescribed probability of success. However, one must keep in mind that (25) requires for the number of tables to satisfy

$$\ell > -\frac{1}{2}\ln\left(1 - \bar{\mathrm{R}}_{\mathrm{ps}}\right). \tag{28}$$

That is, to achieve a given success probability $\bar{\mathrm{R}}_{\mathrm{ps}}$, the number of tables one must use is lower bound by (28). No set of parameters that uses a smaller number of tables can achieve the desired success probability.

Using Proposition 6, we can restate the probability of success (26) as follows.

Proposition 7 *The success probability of the perfect rainbow tradeoff is*

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

Acquiring the online efficiency of the perfect rainbow tradeoff from existing works is also straightforward.

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

$$\begin{aligned}\bar{\mathrm{R}}_{\mathrm{tc}} = &\left(\bar{\mathrm{R}}_{\mathrm{msc}}\ell - \frac{\bar{\mathrm{R}}_{\mathrm{msc}}}{2} + \ell - 2 + \frac{3}{2\ell}\right)\\ &- \left(\frac{\bar{\mathrm{R}}_{\mathrm{msc}}^2\ell}{4} + \bar{\mathrm{R}}_{\mathrm{msc}}\ell^2 - \bar{\mathrm{R}}_{\mathrm{msc}}\ell + \bar{\mathrm{R}}_{\mathrm{msc}} + \ell - 2 + \frac{3}{2\ell}\right)e^{-\bar{\mathrm{R}}_{\mathrm{msc}}\ell}.\end{aligned}$$

Proof According to [13], the expected number of one-way function iterations required to generate the online chain is

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

and that required to resolve alarms is[3]

$$\left[\left\{\bar{\mathrm{R}}_{\mathrm{msc}}\left(\ell - \frac{1}{2}\right) - \left(2 - \frac{3}{2\ell}\right)\right\} + \left\{\left(2 - \frac{3}{2\ell}\right) + \bar{\mathrm{R}}_{\mathrm{msc}}(\ell - 1) - \frac{\bar{\mathrm{R}}_{\mathrm{msc}}^2\ell}{4}\right\}e^{-\bar{\mathrm{R}}_{\mathrm{msc}}\ell}\right]\left(\frac{t}{\bar{\mathrm{R}}_{\mathrm{msc}}\ell}\right)^2.$$

These expected values take the possibility of premature exit from the online phase after discovery of the correct answer into account. The sum of these two terms is the time complexity T.

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. □

[3] The single $e^{c\mathrm{R}}$ appearing in [13, p.312] should be corrected to $e^{c\mathrm{R}\ell}$.

4.2 Storage optimization

As was with the perfect DP tradeoff, storage of a single starting point for the perfect rainbow tradeoff requires $\log m_0$ bits, and (24) shows how this compares with $\log m$. However, unlike the DP case, since $\bar{\mathrm{R}}_{\mathrm{msc}}$ may take values that are very close to 2, there remains the possibility of $\log m_0$ being much larger than $\log m$.

A hint for resolving this problem comes from the derivation process of (25), which shows that $\bar{\mathrm{R}}_{\mathrm{msc}}$ being close to 2 is associated with an unrealistically large amount of pre-computation. In any real-world situation, there will be a bound on the pre-computation cost one is willing to accept. So, let us combine Proposition 6 and (27), and consider a somewhat arbitrary bound of

$$\bar{\mathrm{R}}_{\mathrm{pc}} = \frac{2}{2 - \bar{\mathrm{R}}_{\mathrm{msc}}} \left\{ -\ln(1 - \bar{\mathrm{R}}_{\mathrm{ps}}) \right\} \leq 20, \tag{29}$$

on the pre-computation coefficient. Unless the requirement on the success rate is unrealistically small, this will place a reasonably small bound on the coefficient $\frac{2}{2-\bar{\mathrm{R}}_{\mathrm{msc}}}$ of (24), 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 (29) is that it implies the bound $\bar{\mathrm{R}}_{\mathrm{ps}} \leq 1 - \frac{1}{e^{20}}$ on the success probability one can consider. However, the appearance of a success probability bound is only natural, since a success probability that is arbitrarily close to 1 cannot be achieved without enormous amount of pre-computation. Furthermore, since $99.999999\,\% < 1 - \frac{1}{e^{20}}$, the 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. In the case of the perfect DP tradeoff, truncation was defined only for the DPs, but ending points may take any form with the rainbow tradeoff, so we now consider the truncation of any point from $\mathscr{N}$. We will reuse the terms *truncation map* and *truncated space* that were previously introduced in Sect. 3.2. As was with the DP case, truncation may cause two ending points of a perfect rainbow table to become indistinguishable, and the following translation of Lemma 6 into the language of the rainbow tradeoff solves this problem.

Lemma 10 *Consider a (pre-image uniform) 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* $r\{1 - \exp(-\frac{m}{r})\}$ *distinct truncated points. Conversely, given* $m \leq (1 - e^{-5})r$, *one must truncate* $r \ln\left(\frac{r}{r-m}\right)$ *distinct ending points in order for* m *to be the expected number of distinct truncated points.*

The example values that were given below Lemma 7 remain valid for the perfect rainbow tradeoff. That is, truncation of $1.01596\,m$ 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 $\log r = \epsilon + \log m$ for some small positive integer ϵ.

Analogous to the DP case, if required, one can work with (24) 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 $\bar{\mathrm{R}}_{\mathrm{msc}}$ is sufficiently bounded away from 2, in practice, implies that the non-linearity hidden within $\bar{\mathrm{R}}_{\mathrm{msc}}$ 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 11 *Consider a (pre-image uniform) 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*

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

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

Proof Consider the non-perfect rainbow matrix created with $m_0 = \frac{2m}{2-\bar{\mathrm{R}}_{\mathrm{msc}}}$ 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. Next, consider the online chain created at the i-th 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 may be written as $\prod_{j=0}^{i}\left(1 - \frac{m_{t-j}}{\mathsf{N}}\right)$.

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 match one of the m truncated matrix ending points is $\frac{m}{r}$. Hence, the probability for an online chain of length i to bring about a false alarm associated with the truncation of ending points is

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

Here, one must substitute $\frac{m_i}{\mathsf{N}} = \left(\frac{\mathsf{N}}{m_0} + \frac{i}{2}\right)^{-1}$, which may be found in [2,13], to obtain the equality. This may be approximated and further simplified to

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

where the final equality results from the substitution of m_0, as given by (24).

Now, 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 $\left(1 - \frac{m}{\mathsf{N}}\right)^{\ell(i-1)}$. Since each alarm from and online chain of length i requires $(t - i)$ iterations of the one-way function to resolve, the expected number of extra one-way function iterations induced by truncation-related alarms may be written as

$$\ell\sum_{i=0}^{t}(t-i)\frac{m}{r}\left(1 - \frac{\bar{\mathrm{R}}_{\mathrm{msc}}}{2}\frac{i}{t}\right)^2\left(1 - \frac{m}{\mathsf{N}}\right)^{\ell(i-1)},$$

when the processing of the ℓ rainbow tables are taken into account. Rewriting this in the form

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

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

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

Explicit computation of this definite integral results in our claim. □

Recall that the total online time, without ending point truncation, was given during the proof of Theorem 2. Comparing the complexity with what is given by Lemma 11, it is straightforward to express the effects of ending point truncation on the total online time.

Proposition 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 $r = 2^{\epsilon}m \ll \mathsf{N}$. 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*

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

additional iterations of the one-way function to complete.

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

Let us summarize the issue of storage optimization for the perfect rainbow tradeoff. Each starting point can be stored 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 [14] for the non-perfect rainbow tradeoff, the analysis had to be repeated here for the perfect rainbow tradeoff.

5 Tradeoff algorithm performance comparison

Formulas for the perfect DP and perfect rainbow tradeoffs that give the success rates, online efficiencies, and pre-computation costs in terms of the algorithm parameters were obtained in the previous two sections. Similar formulas for the non-perfect DP and non-perfect rainbow tradeoffs were provided in an earlier work [14]. In this section, we gather all of these information to compare the performances of the four mentioned tradeoff methods.

Our choice to exclude the classical Hellman tradeoff from our comparisons was mainly due to its performance being very similar to that of the non-perfect DP tradeoff. At the scale of the graphs to be given later in this section, curves for the classical Hellman and non-perfect DP tradeoffs would not be clearly distinguishable. One must also be aware of the fact that the classical Hellman tradeoff has a much higher table access frequency than the DP or rainbow tradeoffs, which is an issue that cannot be reflected by the complexities being used in our comparisons.

Below, we will use the symbols $\mathtt{D}_{\mathrm{tc}}$ and $\mathtt{R}_{\mathrm{tc}}$ to denote the tradeoff coefficients, i.e., the $\frac{TM^2}{\mathsf{N}^2}$ values, for the non-perfect DP and non-perfect rainbow tradeoffs, respectively. Also, the symbols $\mathtt{D}_{\mathrm{pc}}$ and $\mathtt{R}_{\mathrm{pc}}$ will denote the number of one-way function iterations expected of the pre-computation phases, counted in multiples of N, for the non-perfect DP and non-perfect rainbow tradeoffs.

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 was first set forth by [14].

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. Our comparisons will be made at the four specific success rate requirements 90, 95, 99, and 99.9 %.

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 four tradeoff methods we are comparing allows for tradeoffs between time and memory of the same $TM^2 = \mathrm{const} \cdot \mathsf{N}^2$ form, 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 that the physical number 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 and 20 bits, respectively, to each pre-computation table entry. Then, to be fair, one must compare the $\frac{100\,TM^2}{\mathsf{N}^2}$ value computed for Method-A against the $\frac{400\,TM^2}{\mathsf{N}^2}$ 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 [14] provided a careful discussion with examples as to how the relative adjustments of the tradeoff coefficients are to be carried out in practice.

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 $\frac{1}{4}\bar{\mathtt{D}}_{\mathrm{tc}}$,

$\bar{\mathrm{R}}_{\mathrm{tc}}$, $\frac{1}{4}\mathrm{D}_{\mathrm{tc}}$, and R_{tc} against each other. At the ends of Sects. 3.2 and 4.2, we had stated that both the perfect DP and perfect rainbow tradeoffs need to allocate "slightly more than $\log m$ bits" to each table entry. We also remarked there that the same was previously shown to be true of the two non-perfect tradeoffs. Now, for both the perfect and non-perfect DP 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 two DP tradeoffs and two rainbow tradeoffs, respectively. In this sense, the two DP tradeoffs 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.

Let us emphasize once more that the accurate adjustment factors can only be fixed by the tradeoff implementer for each specific situation based on the number of bits to be allocated to each table entry and that our performance comparisons to be given below is based on an extreme simplification. We acknowledge that, mainly due to the "slightly more than" part, our choice of the adjustments is slightly in favor of the two DP tradeoffs than what would be seen in practice. For example, it could be that taking $0.3\bar{\mathrm{D}}_{\mathrm{tc}}$, $\bar{\mathrm{R}}_{\mathrm{tc}}$, $0.3\mathrm{D}_{\mathrm{tc}}$, and R_{tc} as the adjusted tradeoff coefficients is more appropriate. However, our later comparisons will indicated that the rainbow tradeoffs are advantageous over the DP tradeoffs, in spite of the undue advantage we are giving to the DP tradeoffs.

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. In other words, one can expect an upper level tradeoff between pre-computation effort and online efficiency. However, unlike the time memory tradeoffs, the upper level tradeoffs exhibited by the four tradeoff methods cannot be expressed with equations having a common form, making it difficult to capture the upper level tradeoff in a certain new coefficient. 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 Performance comparison

Since we are going to compare the tradeoff methods at a few common fixed probabilities of success, the symbols $\bar{\mathrm{D}}_{\mathrm{ps}}$ and $\bar{\mathrm{R}}_{\mathrm{ps}}$ will now be treated as fixed constants.

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

The perfect DP tradeoff will be treated first. It is easy to derive

$$\bar{\mathrm{D}}_{\mathrm{pc}} = \frac{2 + \bar{\mathrm{D}}_{\mathrm{msc}}}{2\,\bar{\mathrm{D}}_{\mathrm{cr}}}\big\{ -\ln(1 - \bar{\mathrm{D}}_{\mathrm{ps}})\big\} \tag{30}$$

from Proposition 2, and since Proposition 3 expresses $\bar{\mathrm{D}}_{\mathrm{cr}}$ as a function of $\bar{\mathrm{D}}_{\mathrm{msc}}$, the pre-computation coefficient can be seen as a function of the single parameter $\bar{\mathrm{D}}_{\mathrm{msc}}$, when $\bar{\mathrm{D}}_{\mathrm{ps}}$ is treated as a constant. Similarly, the combination of Theorem 1 and Proposition 3 expresses the tradeoff coefficient $\bar{\mathrm{D}}_{\mathrm{tc}}$ as a function of the single parameter $\bar{\mathrm{D}}_{\mathrm{msc}}$. Thus the pc-tc curve for the perfect DP tradeoff may be drawn as a curve parameterized by $\bar{\mathrm{D}}_{\mathrm{msc}}$.

It is important to understand that, even when the success rate $\bar{\mathrm{D}}_{\mathrm{ps}}$ and curve parameter $\bar{\mathrm{D}}_{\mathrm{msc}}$ are fixed to specific values, there still remains a single degree of freedom concerning the tradeoff algorithm parameters m, t, and ℓ, 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. The $\bar{\mathrm{D}}_{\mathrm{tc}}$ value appearing with each $(\bar{\mathrm{D}}_{\mathrm{pc}}, \bar{\mathrm{D}}_{\mathrm{tc}})$-pair contained in the pc-tc curve represents an online efficiency with a fully operational time memory tradeoff opportunity.

To be more concrete, suppose that one is given specific $\bar{\mathrm{D}}_{\mathrm{ps}}$ and $\bar{\mathrm{D}}_{\mathrm{msc}}$ values, together with any T and M that satisfy the tradeoff curve of Theorem 1, where the tradeoff coefficient $\bar{\mathrm{D}}_{\mathrm{tc}}$ has been computed from the given $\bar{\mathrm{D}}_{\mathrm{ps}}$ and $\bar{\mathrm{D}}_{\mathrm{msc}}$ values. Then it is easy to check that the sequentially defined parameter set

$$t = \left\{ \frac{\bar{\mathrm{D}}_{\mathrm{msc}}\bar{\mathrm{D}}_{\mathrm{cr}}}{\bar{\mathrm{D}}_{\mathrm{ps}}} \left(1 + \frac{1 + 0.577\bar{\mathrm{D}}_{\mathrm{msc}}}{1 + 0.451\bar{\mathrm{D}}_{\mathrm{msc}}} \frac{\bar{\mathrm{D}}_{\mathrm{msc}}}{1 + \bar{\mathrm{D}}_{\mathrm{msc}}} \right)^{-1} T \right\}^{\frac{1}{2}}, \tag{31}$$

$$m = \frac{\bar{\mathrm{D}}_{\mathrm{msc}}\mathsf{N}}{t^2}, \tag{32}$$

$$\ell = \frac{\mathsf{N}}{mt\bar{\mathrm{D}}_{\mathrm{cr}}}\big\{ -\ln(1 - \bar{\mathrm{D}}_{\mathrm{ps}})\big\} \tag{33}$$

satisfies the four requests or restrictions on $\bar{\mathrm{D}}_{\mathrm{ps}}$, $\bar{\mathrm{D}}_{\mathrm{msc}}$, T, and M. The equivalence of (33) and (10) implies that the success rate $\bar{\mathrm{D}}_{\mathrm{ps}}$ will be achieved with these parameters, while (32) ensures that the given $\bar{\mathrm{D}}_{\mathrm{msc}}$ value is adhered to. Furthermore, since (31) and the first equation in the proof of Theorem 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.

Let us now explain how the pc-tc curve for the the perfect rainbow tradeoff may be plotted. One can combine (27) and Proposition 6 to express the pre-computation coefficient

$$\bar{\mathrm{R}}_{\mathrm{pc}} = \frac{2}{2 - \bar{\mathrm{R}}_{\mathrm{msc}}}\big\{ -\ln(1 - \bar{\mathrm{R}}_{\mathrm{ps}})\big\} = -\frac{2\ell\ln(1 - \bar{\mathrm{R}}_{\mathrm{ps}})}{2\ell + \ln(1 - \bar{\mathrm{R}}_{\mathrm{ps}})} \tag{34}$$

as a function of the single variable ℓ, when $\bar{\mathrm{R}}_{\mathrm{ps}}$ is treated as a fixed constant. Similarly, the substitution of (27) into the formula of Theorem 2 results in an expression for $\bar{\mathrm{R}}_{\mathrm{tc}}$ 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 $\bar{\mathrm{R}}_{\mathrm{ps}}$ and ℓ.

The pc–tc curves for the perfect and non-perfect DP and rainbow tradeoffs are given in Fig. 4 for some specific success rates. Results from the previous work [14] were used to plot the curves for the two non-perfect tradeoffs. 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 box, the (red) empty circles represent the choice of $(\bar{\mathrm{R}}_{\mathrm{pc}}, \bar{\mathrm{R}}_{\mathrm{tc}})$-pairs made available by the perfect rainbow tradeoff, and the (blue) 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 $\ell = \lceil -\frac{1}{2}\ln(1-\bar{\mathrm{R}}_{\mathrm{ps}})\rceil$, as determined by the bound (28). 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 (red) dashed line in each box represents data for the perfect DP tradeoffs. Unless N is small, it is reasonable to treat the parameter $\bar{\mathrm{D}}_{\mathrm{msc}} = \frac{mt^2}{\mathsf{N}}$ 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 $\bar{\mathrm{D}}_{\mathrm{tc}}$ attains its minimum at $\bar{\mathrm{D}}_{\mathrm{msc}} = 1.97255$, regardless of the $\bar{\mathrm{D}}_{\mathrm{ps}}$ value. The lowest point, or the right end, of the dashed curve in each box corresponds to $\bar{\mathrm{D}}_{\mathrm{msc}} = 1.97255$ and the curve is drawn only for the parameter in the range $\bar{\mathrm{D}}_{\mathrm{msc}} \leq 1.97255$. Curve points corresponding to larger $\bar{\mathrm{D}}_{\mathrm{msc}}$ values would be associated with worse online efficiencies at higher pre-computation costs, which would not be used. The (blue) continuous curve in each box represents data for the non-perfect DP tradeoff.

It is quite clear from Fig. 4 that the perfect DP tradeoff (dash) is less desirable than the perfect rainbow tradeoff (circle), regardless of how one wants to balance online efficiency against pre-computation cost. When these normal to high success rates are required, the perfect DP tradeoff is no match for the perfect rainbow tradeoff. 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 interesting fact is that at the 90 % success rate, and possibly also at the 95 % success rate, the perfect DP tradeoff (dash) could be more useful than the non-perfect rainbow tradeoff (dot), unless one is extremely constrained in the amount of pre-computation possible. At the 99 % success rate, even though the perfect DP tradeoff can provide better online efficiency than the non-perfect rainbow tradeoff, the pre-computation penalty seems to be somewhat too high for the small advantage in online efficiency. As for the non-perfect DP tradeoff (line), we can safely say that it is almost obsolete at these high success rate requirements. We also add that if we had included the pc-tc curves for the classical Hellman tradeoff in these graphs, they would have almost overlapped with the curves for the non-perfect DP tradeoff.

The success rates covered by Fig. 4 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 Fig. 4 that the use of the perfect DP tradeoff can be advisable at these less interesting low success rates.

The comments we have given so far concerning Fig. 4 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

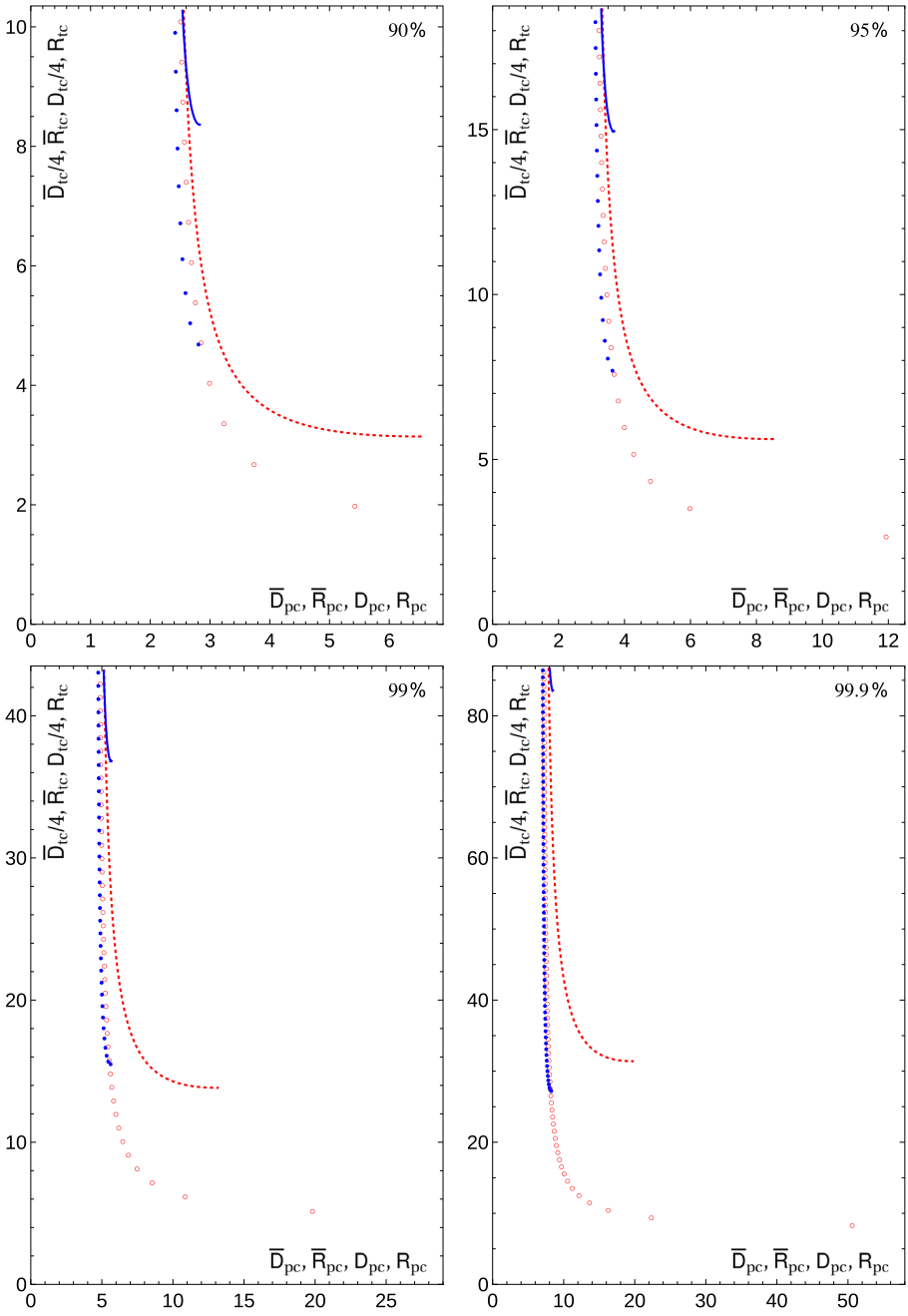

Fig. 4 The tradeoff coefficients $\frac{1}{4}\bar{\mathrm{D}}_{\mathrm{tc}}$ (*dashed*), $\bar{\mathrm{R}}_{\mathrm{tc}}$ (*empty circles*), $\frac{1}{4}\mathrm{D}_{\mathrm{tc}}$ (*line*), and R_{tc} (*dots*), with adjustments suitable for direct comparison, in relation to their respective pre-computation costs, at various success rates

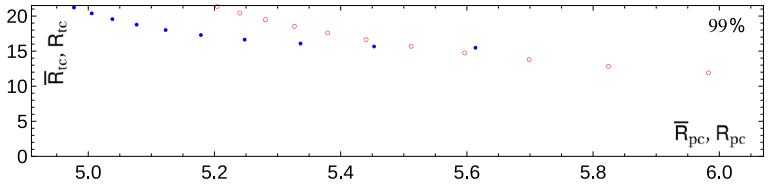

Fig. 5 Tradeoff coefficient for perfect (*circles*) and non-perfect (*dots*) rainbow tradeoffs in relation to their pre-computation costs at 99 % success rate

tradeoff (circle). This is illustrated by Fig. 5, which is an enlarged view of a small rectangular part from the 99 % box of Fig. 4. 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 Fig. 4.

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 perfect DP, non-perfect rainbow, and non-perfect DP 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 perfect DP tradeoff, or even the non-perfect DP tradeoff, favorable at low success rate requirements.

Before ending this section, let us briefly add two remarks concerning the $\bar{D}_{msc} = 1.97255$ value that minimizes the tradeoff coefficient $\bar{D}_{tc}$. Even though our derivation of the formula for $\bar{D}_{tc}$ relied on the empirical result (14), since the range in which (14) is accurate contains the minimizing $\bar{D}_{msc}$ value, one can be confident that $\bar{D}_{msc} = 1.97255$ indeed provides a (local) minimum for $\bar{D}_{tc}$. In the opposite direction, as was already discussed, since $\bar{D}_{msc}$ values greater than 1.97255 will not be used, it suffices to restrict our attention to parameters in the $0 < \bar{D}_{msc} \leq 1.97255$ range, so that our confirmation of (14) being accurate in the range $0 < \bar{D}_{msc} < 2.3$ is more than enough.

6 Conclusion

In this work, we analyzed the execution complexity of the perfect DP tradeoff and computed its tradeoff coefficient. We also combined the existing results on the execution complexity of the perfect rainbow tradeoff to present its online efficiency. Using this information, the performances of the perfect DP tradeoff and perfect rainbow tradeoff were compared against each other. Results from the previous work [14] on the non-perfect table tradeoffs were also included in our comparison for easy reference. Our conclusion, in overly simplified terms, is that the perfect rainbow tradeoff is advantageous over the classical Hellman, non-perfect DP, non-perfect rainbow, and perfect DP tradeoffs.

On the surface, our conclusion may seem to be a repetition of the claim given by the article [21] that first introduced the rainbow tradeoff, but the two claims differ fundamentally in their contents. The previous work [21] measured the storage size in terms of the number of table entries, but the physical storage size was considered in our comparisons. Our comparison was based on the expected execution complexities, rather than the worst case complexities, which were used by [21]. Finally, our comparison considered both the online efficiency and

the cost of pre-computation, whereas only the optimal online behavior was considered by [21]. In other words, different concepts of one algorithm being better than another algorithm were employed by [21] and this work in comparing tradeoff algorithms.

In addition to the conceptual differences which make our comparisons more meaningful in practice than the previous claims of [21], there was also a difference in the accuracy of the two comparisons. The claim of [21] relied on parameter sets for different algorithms that were heuristically shown to bring about roughly corresponding success rates, but the parameter sets for different algorithms were conditioned to achieve exactly the same success rate in our comparison. In addition, our comparison, unlike that of [21], fully accounted for the cost of resolving false alarms, which constitutes a significant portion of the online time.

Despite our simplified conclusion of the perfect rainbow tradeoff outperforming the other four algorithms, we do not rule out the possibility of encountering situations where the conclusions could be different. One example would be when lowering the pre-computation cost is immensely important. There may also be issues we have ignored, such as the possibility of loading a pre-computation table fully into fast memory or table lookup characteristics, which could make the two DP tradeoffs, with smaller individual tables, preferable over the two rainbow tradeoffs. Furthermore, recent implementation platforms such as GPUs, where the scheduling of the online computation needs to be totally different from most computation platforms and accesses to large storage are inconvenient, could call for a completely different analysis of the execution complexities.

It remains to extend this work and verify whether the lesser known recently proposed tradeoff algorithms [1,12,15,20,26–28,31,33,34] are superiority to the more widely known algorithms covered by [14] and this work, in the sense we have considered in this work. Note that the results and approaches of this work have already been used to show [17,18] that the fuzzy rainbow tradeoff [3,4] could be seen as performing better than even the perfect rainbow tradeoff. Also note that [16] had arrived at a negative conclusion concerning the non-perfect table case of the pH tradeff [12,24].

Acknowledgments The authors thank Wenhao Wang of IIE CAS for bringing a critical error that appeared in an earlier version of this paper to our attention. This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (NRF-2012R1A1B4003379). G. W. Lee was also supported by Global Ph.D. Fellowship Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2011-0030901). Part of this work was done while J. Hong was visiting Department of Mathematics, U. C. Davis, and he is grateful for their hospitality.

Appendix 1: Effects of chain length bounds and parallelization on the DP tradeoff

Let us briefly comment on the two techniques for the DP tradeoff that were not considered during our complexity analyses. Specifically, these were the use of upper and lower bounds on the chain lengths and the parallel processing of tables during the online phase.

The use of chain length bounds will increase the number of pre-computation chains that are discarded, so that the amount of pre-computation must be increased in order to maintain the success rate. The effect of chain length bounds on the online efficiency is unclear, but making the pre-computation chain lengths shorter has the tendency to reduce chain merges and false alarms, so this could have a positive effect. However, a quick review of Fig. 4 shows that no small enhancement in the online efficiency is likely to make the perfect DP tradeoff more preferable over the perfect rainbow tradeoff, even if no penalty on the pre-computation

cost was involved. Hence, there seems to be little reason to consider the perfect DP tradeoff variant that utilizes chain length bounds, except possibly at low success rates.

The work [12,24] 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. However, the recent analysis on the non-perfect parallel DP tradeoff [16] indicates that one cannot hope to see any drastic improvement of the perfect DP tradeoff performance through parallelization. For now, there seems to be no reason to expect the parallel perfect DP tradeoff to perform better than the perfect rainbow tradeoff, especially under the high success rate requirements which are of interest.

Appendix 2: Appropriate chain length upper bound

Let us analyze how often one can expect to see over-length chains when an upper bound on the chain length is set. This information will give us an indication of how large a bound qualifies as a *sufficiently large* chain length bound.

We first wish to write down the probability for a chain generated from a given starting point to fall into an infinite loop without reaching a DP within its first $\hat{t}$ iterations. The chain can fall into a loop if the first iteration lands on the given starting point itself. The probability for such an event can be stated to be $\frac{1}{\mathsf{N}}$, when the chain is treated as a random walk. If the first iteration lands on a non-DP that is different from the starting point, which happens with probability $1-\frac{1}{t}-\frac{1}{\mathsf{N}}$, then the chain can fall into a loop if the second iteration lands on either of the first two points, which happens with probability $\frac{2}{\mathsf{N}}$. Continuing, the probability in question may be written as

$$\begin{aligned}\frac{1}{\mathsf{N}}+\left(1-\frac{1}{t}-\frac{1}{\mathsf{N}}\right)\frac{2}{\mathsf{N}}+\left(1-\frac{1}{t}-\frac{1}{\mathsf{N}}\right)\left(1-\frac{1}{t}-\frac{2}{\mathsf{N}}\right)\frac{3}{\mathsf{N}}+\cdots\cdots\\ \cdots\cdots+\left(1-\frac{1}{t}-\frac{1}{\mathsf{N}}\right)\left(1-\frac{1}{t}-\frac{2}{\mathsf{N}}\right)\cdots\left(1-\frac{1}{t}-\frac{\hat{t}-2}{\mathsf{N}}\right)\frac{\hat{t}-1}{\mathsf{N}}.\end{aligned} \tag{35}$$

If we assume $t\hat{t} \ll \mathsf{N}$, the $\frac{i}{\mathsf{N}}$ term appearing inside each set of parentheses can be ignored and the above may be approximated by

$$\frac{1}{\mathsf{N}}+\left(1-\frac{1}{t}\right)\frac{2}{\mathsf{N}}+\cdots+\left(1-\frac{1}{t}\right)^{\hat{t}-2}\frac{\hat{t}-1}{\mathsf{N}}. \tag{36}$$

Arguing as before, we can state that the probability for a chain not to reach a DP within its first $\hat{t}$ iterations, without falling into a loop, is

$$\left(1-\frac{1}{t}-\frac{1}{\mathsf{N}}\right)\left(1-\frac{1}{t}-\frac{2}{\mathsf{N}}\right)\cdots\left(1-\frac{1}{t}-\frac{\hat{t}-2}{\mathsf{N}}\right)\left(1-\frac{1}{t}-\frac{\hat{t}-1}{\mathsf{N}}\right), \tag{37}$$

and this can be approximated by

$$\left(1-\frac{1}{t}\right)^{\hat{t}-1}, \tag{38}$$

under the assumption that $t\hat{t} \ll \mathsf{N}$.

The sum of (36) and (38)

$$\frac{1}{\mathsf{N}}+\left(1-\frac{1}{t}\right)\frac{2}{\mathsf{N}}+\cdots+\left(1-\frac{1}{t}\right)^{\hat{t}-2}\frac{\hat{t}-1}{\mathsf{N}}+\left(1-\frac{1}{t}\right)^{\hat{t}-1}, \tag{39}$$

is the probability for a chain not to reach a DP within its first $\hat{t}$ iterations. If we further assume that $\frac{\hat{t}}{t}$ is not *too* large, we can approximate and rewrite this in the form

$$\frac{t^2}{\mathsf{N}}\left(\frac{1}{t}+e^{-\frac{1}{t}}\frac{2}{t}+e^{-\frac{2}{t}}\frac{3}{t}+\cdots+e^{-\frac{\hat{t}-2}{t}}\frac{\hat{t}-1}{t}\right)\frac{1}{t}+e^{-\frac{\hat{t}-1}{t}}, \tag{40}$$

and approximate the above once more with the definite integral

$$\frac{t^2}{\mathsf{N}}\int_0^{\frac{\hat{t}}{t}} e^{-u}u\,du+e^{-\frac{\hat{t}}{t}}=\frac{t^2}{\mathsf{N}}\left\{1-\left(1+\frac{\hat{t}}{t}\right)e^{-\frac{\hat{t}}{t}}\right\}+e^{-\frac{\hat{t}}{t}}. \tag{41}$$

This probability approaches $\frac{t^2}{\mathsf{N}}=O\left(\frac{1}{m}\right)$ very quickly, as $\frac{\hat{t}}{t}$ is increased. For example, even at the moderately large chain length bound of $\hat{t}=15\,t$, the probability for a randomly generated chain to be discarded due to its length is $\frac{t^2}{\mathsf{N}}0.999995+3.05902\times10^{-7}$, which is small enough for our purposes.

As a word of caution, we add that, due to the first term of $O\left(\frac{1}{m}\right)$ order, the number of long chains expected during the generation of a full DP matrix cannot be made arbitrarily close to zero by increasing the chain length bound, since each DP matrix contains m chains. Only the fraction of over-length chains among all chains approaches zero with the increase of the chain length bound.

Appendix 3: Rigorous proof of Lemma 3

The very short proof of Lemma 3 given in the main body of this paper will seem sufficient to most cryptographers. However, there are subtle issues involving random functions that were ignored in the proof. This section is an attempt at resolving these issues. We strongly urge any interested reader to review [14, Appendix B] before reading this section.

Recall that we are using $\mathscr{D}_k(F)$ to denote the set of elements of $\mathscr{N}$ that are k-many F-iterations away from their closest DPs, and suppose that D_0, D_1, …, D_k are any collection of mutually disjoint subsets of $\mathscr{N}$, with D_0 denoting the set of all DPs. Let us consider any function $F:\mathscr{N}\to\mathscr{N}$ and discuss when the series of conditions

C1: $D_i=\mathscr{D}_i(F)$, for $i=1,\ldots,k$

would be satisfied by the function. Note that the omitted condition $D_0=\mathscr{D}_0(F)$ is always satisfied.

The series of conditions C1 is satisfied by the function F if and only if the following three items are all satisfied:

C2: $D_i=\mathscr{D}_i(F)$, for $i=1,\ldots,k-1$.
C3: $F(D_k)\subset D_{k-1}$.
C4: $F(\mathscr{N}\setminus D_k)\cap D_{k-1}=\emptyset$.

Repeating this argument, we see that the satisfaction of Condition-C1 by an F is equivalent to the satisfaction of the following two series of conditions.

C5: $F(D_k) \subset D_{k-1}$, ..., $F(D_2) \subset D_1$, and $F(D_1) \subset D_0$.
C6: $F(\mathscr{N} \setminus D_k) \cap D_{k-1} = \emptyset$, ..., $F(\mathscr{N} \setminus D_2) \cap D_1 = \emptyset$, and $F(\mathscr{N} \setminus D_1) \cap D_0 = \emptyset$.

Suppose that we are now trying to *construct* a function that satisfies these conditions. Then the condition $F(D_k) \subset D_{k-1}$ would certainly places a restriction on how we may define F on D_k. However, since the set D_k does not overlap with any of the other sets $D_0, \ldots, D_{k-1}$, none of the other sub-conditions of C5 places any restriction on the definition of F on D_k. Next, the sub-condition $F(\mathscr{N} \setminus D_k) \cap D_{k-1} = \emptyset$ of C6 is certainly irrelevant to the definition of F on D_k. The other sub-conditions of C6 are not quite so irrelevant, but since D_{k-1} does not overlap with any of the sets $D_0, \ldots, D_{k-2}$, as long as the condition $F(D_k) \subset D_{k-1}$ is adhered to, they are automatically satisfied and do not places any additional restriction of how we may define F on D_k. More generally, for each $i = 1, \ldots, k$, the condition $F(D_i) \subset D_{i-1}$ is the only restriction placed on the definition of F on D_i by C5 and C6. In other words, asking for C1 to be satisfied by a function F places no restriction on the definition of F on D_i other than that $F(D_i) \subset D_{i-1}$ be satisfied.

The discussion given so far can be reinterpreted as follows. Let $D_0, D_1, \ldots, D_k$ be any collection of mutually disjoint subsets of $\mathscr{N}$, with D_0 denoting the set of all DPs. By choosing a function $F : \mathscr{N} \to \mathscr{N}$ satisfying Condition-C1, one determines functions $F_i : D_i \to D_{i-1}$ for each $i = 1, \ldots, k$. Let us fix any i. If the choice of F is made uniformly at random from the set of all function satisfying Condition-C1, the distribution of the corresponding $F_i : D_i \to D_{i-1}$, defined on the set of all functions having domain D_i and co-domain D_{i-1}, also becomes the uniform distribution.

We are now ready to prove the lemma. Choose any explicit family of disjoint sets D_0, ..., D_k such that D_0 is the set of all DPs. For a random function satisfying Condition-C1, since each $F_i : D_i \to D_{i-1}$ is a random function, we can expect to see

$$\frac{|F(D')|}{|D_{i-1}|} = 1 - \exp\left(-\frac{|D'|}{|D_{i-1}|}\right) \tag{42}$$

for every set $D' \subset D_i$ and $i = 1, \ldots, k$. So far, we have fixed $D_0, \ldots, D_k$ first and have claimed information on iterated image sizes only for functions F satisfying Condition-C1, which is associated with the sets $D_0, \ldots, D_k$. However, observe that this iterative equation depends only on the set sizes $|D'|, |D_0|, \ldots, |D_k|$ and not on the specific sets $D', D_0, \ldots, D_k$. Also recall that (11) is expected of a random function. Interpreting this as (11) holding accurately for the vast majority of functions F, the truth of Lemma 3 follows. A review of [14, Appendix B] is required to fully understand this final statement.

Appendix 4: Optimal rainbow tradeoff parameters

In this section we discuss the parameter set for the perfect rainbow tradeoff that the previous analysis [2] suggested and claimed to be optimal. Let us explain the steps they take to fix the parameters, in the language of this work. The required success rate $\bar{\mathrm{R}}_{\mathrm{ps}}$ and the total number of table entries M are treated as the externally supplied parameters. First (28) is used to fix the number of tables $\ell = \lceil -\frac{1}{2} \ln(1 - \bar{\mathrm{R}}_{\mathrm{ps}}) \rceil$. This forces the number of starting points per table to $m = \frac{M}{\ell}$ and we know from (26) that the remaining parameter t must be fixed to

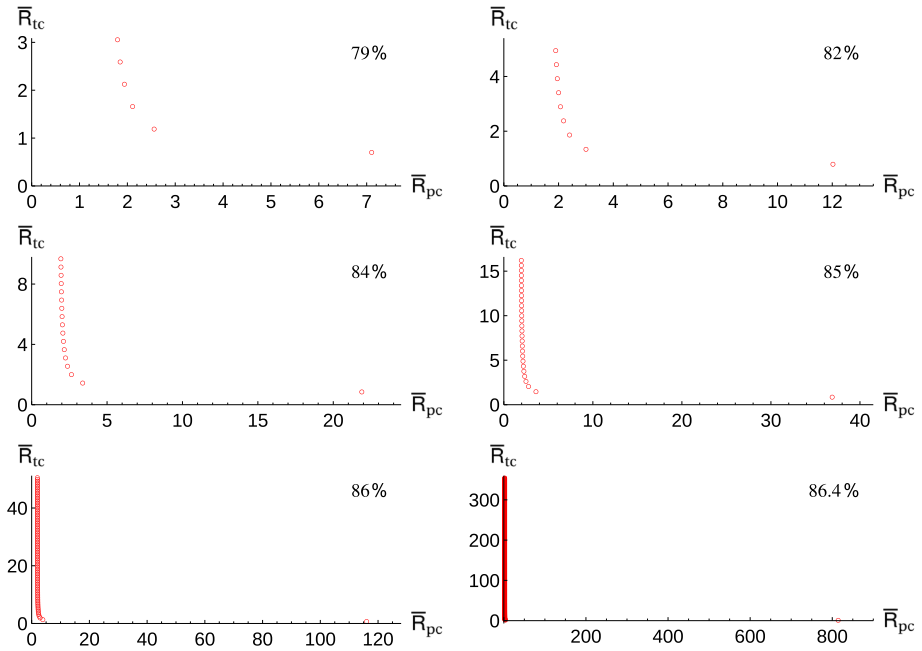

Fig. 6 Pre-computation cost versus tradeoff coefficient for the perfect rainbow tradeoff at success rates slightly lower than $1 - \frac{1}{e^2} = 86.466\ \%$

$$t = -\frac{\mathsf{N}}{m\ell} \ln(1 - \bar{\mathsf{R}}_{\mathrm{ps}}) = -\frac{\mathsf{N}}{M} \ln(1 - \bar{\mathsf{R}}_{\mathrm{ps}}), \tag{43}$$

if the requested success rate is to be achieved. In short, the number of tables is chosen to be as small as possible, subject to the condition (28), and the rest of the parameters are set to what they must be in order to satisfy the externally given requirements.

It is not hard to show that the tradeoff coefficient $\bar{\mathsf{R}}_{\mathrm{tc}}$ of Theorem 2, when combined with (27), is an increasing function of ℓ in the range $\ell \geq 1$ and $0 \leq \bar{\mathsf{R}}_{\mathrm{ps}} < 1$. This implies that the parameters suggested by [2] are optimal in the sense that it gives the smallest possible tradeoff coefficient among those achieving the intended success rate. An easier approach would be to state that, the optimal parameter set of [2] corresponds to the rightmost empty circle in each box of Fig. 4, which is clearly the lowest dot in each box.

However, we want to emphasize that the rightmost circle of each box is just one of the many options that are available to the tradeoff algorithm implementer, and that this specific option is not likely to be favored over others when the pre-computation cost is taken into account, except possibly at low success rates. Furthermore, Fig. 6 demonstrates that there are success rates for which using parameters corresponding to the rightmost dot is certainly impractical.

The parameter set that is most efficient in terms of the online resource requirements is very often not the most practical or reasonable parameter set option among those made available by a tradeoff method. The current work provides the information which enables the tradeoff implementer to decide on the most sensible, rather than the most online-efficient, parameter to use.

References

1. Ågren M., Johansson T., Hell M.: Improving the rainbow attack by reusing colours. CANS 2009. LNCS, vol. 5888, pp. 362–378. Springer, Berlin (2009).
2. Avoine G., Junod P., Oechslin P.: Characterization and improvement of time-memory trade-off based on perfect tables. ACM Trans. Inf. Syst. Secur. **11**(4), 17:1–17:22 (2008). Preliminary version presented at INDOCRYPT 2005.
3. Barkan E.P.: Cryptanalysis of Ciphers and Protocols. Ph.D. Thesis, Technion—Israel Institute of Technology, Haifa (2006).
4. Barkan E., Biham E., Shamir A.: Rigorous bounds on cryptanalytic time/memory tradeoffs. Advances in Cryptology—CRYPTO 2006. LNCS, vol. 4117, pp. 1–21. Springer, Heidelberg (2006).
5. Biryukov A., Shamir A.: Cryptanalytic time/memory/data tradeoffs for stream ciphers. In: Advances in Cryptology—ASIACRYPT 2000. LNCS, vol. 1976, pp. 1–13. Springer, Heidelberg (2000).
6. Biryukov A., Shamir A., Wagner D.: Real time cryptanalysis of A5/1 on a PC. In: FSE 2000, LNCS, vol. 1978, pp. 1–18. Springer, Heidelberg (2001).
7. Borst J.: Block Ciphers: Design, Analysis, and Side-Channel Analysis. Ph.D. Thesis, Katholieke Universiteit Leuven, Leuven (2001).
8. Borst J., Preneel B., Vandewalle J.: On the time-memory tradeoff between exhaustive key search and table precomputation. In: Proceedings of the 19th Symposium on Information Theory in the Benelux, pp. 111–118. WIC (1998).
9. Denning D.E.: Cryptography and Data Security. Addison-Wesley, Reading (1982).
10. Fiat A., Naor M.: Rigorous time/space tradeoffs for inverting functions. In: Proceedings of the twenty-third annual ACM Symposium on Theory of Computing, pp. 534–541. ACM, New York (1991).
11. Hellman M.E.: A cryptanalytic time-memory trade-off. IEEE Trans. Inf. Theory **26**, 401–406 (1980).
12. Hoch Y.Z.: Security analysis of generic iterated hash functions. Ph.D. Thesis, Weizmann Institute of Science, Rehovot (2009).
13. Hong J.: The cost of false alarms in Hellman and rainbow tradeoffs. Des. Codes Cryptogr. **57**(3), 293–327 (2010).
14. Hong J., Moon S.: A comparison of cryptanalytic tradeoff algorithms. J. Cryptol. **26**(4), 559–637 (2013).
15. Hong J., Jeong K.C., Kwon E.Y., Lee I.-S., Ma D.: Variants of the distinguished point method for cryptanalytic time memory trade-offs. In: ISPEC 2008. LNCS, vol. 4991, pp. 131–145. Springer, Heidelberg (2008).
16. Hong J., Lee G.W., Ma D.: Analysis of the parallel distinguished point tradeoff. In: Progress in Cryptology—INDOCRYPT 2011. LNCS, vol. 7107, pp. 161–180. Springer, Heidelberg (2011).
17. Kim B.-I., Hong J.: Analysis of the non-perfect table fuzzy rainbow tradeoff. In: ACISP 2013. LNCS, vol. 7959, pp. 347–362. Springer, Heidelberg (2013).
18. Kim B.-I., Hong J.: Analysis of the perfect table fuzzy rainbow tradeoff. J. Appl. Math. **2014**, 765394 (2014). doi:10.1155/2014/765394.
19. Kim J.W., Seo J., Hong J., Park K., Kim S.-R.: High-speed parallel implementations of the rainbow method based on perfect tables in a heterogeneous system. Softw. Pract. Exper. **45**(6), 837–855 (2015).
20. Mukhopadhyay S., Sarkar P.: Nearly orthogonal rainbow tables. Indian Statistical Institute Technical Report, No. ASD/2004/9, November (2004).
21. Oechslin P.: Making a faster cryptanalytic time-memory trade-off. In: Advances in Cryptology—CRYPTO 2003, LNCS, vol. 2729, pp. 617–630. Springer, Heidelberg (2003).
22. Quisquater J.-J., Stern J.: Time-Memory Tradeoff Revisited. Unpublished, December (1998).
23. Saran N.: Time Memory Trade Off Attack on Symmetric Ciphers. Ph.D. Thesis, Middle East Technical University, Ankara (2009).
24. Shamir A.: Random Graphs in Security and Privacy. Invited talk at ICITS (2009).
25. Standaert F.-X., Rouvroy G., Quisquater J.-J., Legat J.-D.: A time-memory tradeoff using distinguished points: New analysis & FPGA results. In: Cryptographic Hardware and Embedded Systems—CHES 2002, LNCS, vol. 2523, pp. 593–609. Springer, Heidelberg (2003).
26. Thing V.: Virtual expansion of rainbow tables. In: Advances in Digital Forensics VI, pp. 243–256. IFIP AICT 337 (2010).
27. Thing V.L.L., Ying H.-M.: A novel time-memory trade-off method for password recovery. Digital Investigation **6**, S114–S120 (2009).
28. Thing V.L.L., Ying H.-M.: Rainbow table optimization for password recovery. Int. J. Adv. Softw. **4**, 479–488 (2011).
29. Tomašević V., Tomašević M.: An analysis of chain characteristics in the cryptanalytic TMTO method. Theor. Comput. Sci. **501**, 52–61 (2013).

30. Wang W., Lin D.: Analysis of multiple checkpoints in non-perfect and perfect rainbow tradeoff revisited. In: ICICS 2013. LNCS, vol. 8233, pp. 288–301. Springer, Heidelberg (2013).
31. Wang W., Lin D., Li Z., Wang T.: Improvement and analysis of VDP method in time/memory tradeoff applications. In: ICICS 2011. LNCS, vol. 7043, pp. 282–296. Springer, Heidelberg (2011).
32. Wiener M.J.: The full cost of cryptanalytic attacks. J. Cryptol. **17**(2), 105–124 (2004).
33. Ying H.-M., Thing V.L.L.: A novel rainbow table sorting method. CYBERLAWS **2011**, 35–40 (2011).
34. Zhang W., Zhang M., Liu Y., Wang R.: A new time-memory-resource trade-off method for password recovery. In: ICCIIS 2010, pp. 75–79. IEEE, New York (2010).